Yannis Haralambous

A Course in Natural Language Processing

Springer

A Course in Natural Language Processing

Yannis Haralambous

A Course in Natural Language Processing

Springer

Yannis Haralambous
Lab-STICC, UMR CNRS 6285
IMT Atlantique
Brest, France

ISBN 978-3-031-27225-7 ISBN 978-3-031-27226-4 (eBook)
https://doi.org/10.1007/978-3-031-27226-4

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

To my wife Tereza
and my daughters Ernestine and Danaé,
for their love and patience.

Preface

From ELIZA to ChatGPT

Between 1966 and 2022, a miracle happened. 1966 was the year of the release of ELIZA, a chatbot that imitated a psychotherapist and gave the impression of leading ☞ 253 a conversation with its human interlocutor. When the human said, “I hate my work,” ELIZA would say, “Tell me more…,” when the human said, “I eat tons of chocolate per day,” ELIZA would say, “I think you are blocking what you really want to say,” without ever really broaching the subject. The illusion was good, but it was definitely an illusion.

In 2022, ChatGPT was released. ChatGPT knows everything, has advice on everything, and can interact with humans in a perfect dialogue. Not only does it pass the Turing test, but after a few minutes, the human has completely forgotten that she is communicating with a machine. Humanity has never been so close to the technological singularity.

What happened between 1966 and 2022? When did the miracle occur?

In fact, there was no miracle, just hard work by thousands of people working in a discipline called Natural Language Processing or Computational Linguistics, depending on the researcher’s academic background. This book aims to introduce the reader to it.

Natural Language Processing (NLP) is at the crossroads of many disciplines: linguistics, computer science, artificial intelligence, cognitive psychology, and mathematics, to name only a few. There are many books available on the subject. Every book on NLP chooses its topics among these disciplines according to its specific goals and expectations. Some books focus more on mathematics, others more on computer science, and others on linguistics. Where does this book stand in relation to its predecessors?

We live in an era of transformers and embeddings, of neural architectures that accomplish fabulous things. But however fabulous these tools are, one needs to know when and why to apply them. The first step to finding meaningful solutions to a problem is to understand the problem thoroughly.

This is the aim of this book. To shed light on the beautiful complexity of language. Part of this beautiful complexity consists of the fact that there are many theories, many layers, and many ways to approach language, sometimes complementary, sometimes conflicting. Some approaches to language are purely theoretical, and others involve mathematical theories or computer tools. No matter what the approach, the goal has always been to understand how language works. Theories abound, and no theory is perfect, but every theory is an opportunity to get insight into some aspect of language, knowledge, and ourselves. One needs a good technical knowledge of neural networks and a solid insight into language to apply deep learning architectures to linguistic data successfully. This book aims to give the reader insight into language, its theories, and its tools.

Pedagogical Objectives

> A good attitude to the preparation of written mathematical exposition is to pretend that it is spoken. Pretend that you are explaining the subject to a friend on a long walk in the woods, with no paper available.
>
> Paul R. Halmos, *How to Write Mathematics*

The book the reader holds in her hands is entitled “A *Course* in Natural Language Processing,” so it has particular pedagogical objectives. The most important is *simplicity*: a concept must be explained simply—otherwise, it is better not to mention it. The reader is not there to unravel the tortuous threads of the author’s thought. She’s there to learn.

Today, information is everywhere. When it comes to theories and methods, seminal works exist, written by the instigators of the theories or the inventors of the methods. What can be better than that, as it is learning directly from the source? They are often intended for insiders, readers with prior field experience and knowing the ins and outs. It’s up to the educator to make linguistic theories accessible.

With his proverbial sense of humor, Einstein used to say, “Everything should be made as simple as possible, but *no simpler*.” By this, he meant that oversimplification leads to caricature and misinformation. When a difficulty is inherent in a field, it must be faced, not avoided, by hiding it from the learner. If the difficulty is unavoidable, it exists because of a property of the object of study. And so, the learner will come up against it sooner or later. So, it’s vital to have the means to deal with it effectively.

It won’t take long for the reader to realize that this work has a distinctive style. It is written in a language that is sometimes personal, sometimes formal. Often with cultural or encyclopedic information forming little digressions. This is no accident. The author is convinced that this little extra doesn’t slow learning but facilitates it.

To give an example, in the chapter on logic and the famous syllogism “All humans are mortal, Socrates is a human; therefore he is mortal,” the author provides the reader with the information, backed up by references, that immortality is indeed an issue since there actually exists an immortal living organism, the jellyfish t.dohrnii. ☞ 279 This information on the biology of jellyfish has nothing to do with NLP. But it does help to put things into perspective, to stand back. And it is perspective that the learner needs, above all. She builds up her knowledge by sorting information into what is essential for NLP and what is incidental. She trains herself to separate the wheat from the chaff to retain the essentials by putting them into a global context.

Another feature of this book is the almost constant presence of science fiction (notably the Dr. Who series). This is because science fiction opens the mind, offering alternatives to thought. It deals with real issues of our world and the world to come, but it does it through fiction, sparing the reader with the gravity of current events and the constant stress of the “dark future coming ahead of us.” Science fiction can shake up certainties and stimulate the imagination while maintaining a coherent narrative framework. The Dr. Who series has been around for sixty years, accompanying British society through all its mutations and constituting the secret garden of several generations. Its choice, strictly limited to examples, makes them more personal and, therefore, more accessible.

For Whom Is This Book Written, and How To Read It

This book is based on a Natural Language Processing course taught since 2010 at IMT Atlantique, a French engineering school (“grande école d’ingénieurs”$_{\mathrm{FR}}$), located in Brest, Brittany. Students attending the course belong to the 3rd year of the regular engineering track (2nd year graduates) and to a Master of Science called “Data Science.” Therefore, technically, the book should be aimed at graduate students. Nevertheless, the prerequisites are merely high-school set theory, algebra, analysis, and some notions of probability, so it should be accessible to undergraduates, as every concept beyond that level is thoroughly explained.

There is no linear way of reading this book. Part One presents the fundamental notions of linguistics. Not every theory or tool presented in this part will be helpful to every reader. Still, it is good to know these approaches exist so the reader can apply them whenever needed. Chapter 2 is a bit special since grapholinguistics is a very young discipline, and many mainstream linguists consider written language as a mere epiphenomenon of spoken language and, therefore, unworthy to be studied. The author has given Chapter 2 extra care because it is one of his favorite research axes, but also because its material is not provided in any other textbook. Part Two presents the kind of mathematics one uses in NLP (and computer science, more generally). Graphs, in particular, are essential in all fields of science, so Chapter 7 is a must-read. Formal languages and logic are the ingredients of the symbolic approach to language (and Chapter 6 is strongly connected to the symbolic approach, as exercises show clearly). Chapter 10 is about a popular technology, the Semantic Web, which is best understood when read after, or in parallel, with Chapter 9. Part Three is a bit more technical. It deals with tools everyone is supposed to know about yet are not taught anywhere. Finally, Part Four presents statistical methods, starting from Ancient Greece and ending with ChatGPT. Depending on the kind of NLP the reader plans to do (symbolic or statistical), she would focus more on Part Four or Part Two. Nevertheless, it is our firm belief that every user of NLP, whether she uses more symbolic or statistical methods, must have a global vision of the fundamentals of Linguistics, presented in Part One.

Acknowledgments

My gratitude goes, above all, to my wife, Tereza, and my daughters, Ernestine and Danaé, for their love and patience. This book is dedicated to them.

I wish to thank the many people that helped him during the writing of this book: John Esling (the *iPA Phonetics* app), Daniel Harbour (on phonetics), Terry Joyce and Mary Dyson (on psycholinguistics), Egor Dudyrev (the *FCApy* package), Tim Osborne and Guy Perrier (on dependencies), Tian Tian (on Chinese NLP), Nick Nicholas and especially Martin Bays (on Lojban), Norbert Fuchs (on Attempto), Ziyue Xiang (on LaTeX3 regex), Volker Dehs (for his expertise on Jules Verne), Brett Gamboa (for his expertise on doubling in Shakespeare's plays), and Ben Nadel for his enthusiasm on regular expressions.

Lastly, I would like to thank my editor, Alexandru Ciolan, for his professionalism, responsiveness, and always cheerful and creative attitude.

Brest, France,
September 2023

Yannis Haralambous

Contents

Chapter 1
Introduction

1.1 What Is Language in the First Place?

> Language is a system of communication used by humans to express thoughts, ideas, emotions, and information through the use of spoken or written symbols. It is a complex and dynamic system that enables humans to convey meaning, share experiences, and interact with one another.
>
> ChatGPT, May 2023

Linguistics is the science that investigates language. But what is *language*? According to the *Concise Oxford Dictionary of Linguistics* (= CODL) [14], language is

> (1) A language in the ordinary sense: e.g., English or Japanese. Opp. dialect, also as in ordinary usage. (2) The phenomenon of vocal and written communication among human beings generally, again as in ordinary usage.

But this definition is already specific to linguistics since this is the scope of the dictionary [14]. The more general *Merriam-Webster's Dictionary of English* (= MW) [16] provides a more general spectrum of meanings of the word "language," among which we find the following:

> (1-a) the words, their pronunciation, and the methods of combining them used and understood by a community (as in "studies the French *language*")
> (1-b-1): audible, articulate, meaningful sound as produced by the action of the vocal organs
> (1-b-2): a systematic means of communicating ideas or feelings by the use of conventionalized signs, sounds, gestures, or marks having understood meanings (as in "the *language* of mathematics")
> (1-b-4): the means by which animals communicate (as in "the *language* of birds")
> (1-b-5): a formal system of signs and symbols (such as FORTRAN or calculus in logic), including rules for the formation and transformation of admissible expressions
> (1-b-6) the set of symbolic instruction codes, usually in binary form, that is used to represent operations and data in a machine (such as a computer).

It is interesting to compare the two definitions: on the one hand, one could say that CODL's (1) has a problem of auto-referentiality (it defines a language as being…

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_1

a language), which is not present in MW's (1-a) but, then again, (1-a) mentions ☞ 65 *words* which, as we will see, are a very ill-defined notion, and furthermore MW's ☞ 45 (1-a) is *phonocentric*, in the sense that it only deals with spoken language, while CODL's (1) leaves the question of modality open. On the other hand, CODL's (2) is broken into two parts in MW: (1-b-1) gives the means but not the purpose, and (1-b-2) encompasses three modalities (oral, written, and gestural) but widens the scope, including jargons, such as the "language of mathematics."

The title of this book uses the term *natural language*. How does it fit in these definitions? MW defines *natural language* as

> a language that is the native speech of a people,

as opposed to *artificial language* which is

> a language devised by an individual or a small group of individuals and proposed for an international language or some more specific purpose (such as aptitude testing) but not functioning as the native speech of its users.

This draws a clear line between cases such as French (natural language) and Esperanto (artificial language, even though, according to Wikipedia, in 2004, around 2,000 children were growing up speaking Esperanto, hence considering it their native language). But what makes a natural language "natural"?

This is a very important question for many reasons. First of all, it is the *naturalness* of natural language that makes the subject of this book a difficult one. If humans were communicating in formal languages (in the sense of MW's 1-b-5), then Chapters 9–11 of this book would suffice. The communication problem between humans and artificial intelligence would be solved (and some trees would be saved by the drastic reduction of the number of pages of this book). Secondly, as both formal/artificial and natural languages have their pros and contras, there ex- ☞ 197 ists a field in Artificial Intelligence (AI) dealing with *controlled languages*, that is languages intermediate between natural languages and formal languages.

To give a partial answer to the question about the naturalness of language, let us first consider formal languages, such as *programming languages*. When we attempt to master a programming language, we learn its morphology (the keywords and the way to write nominals, such as variable names, or literals, such as character strings), its syntax (the way to combine keywords, nominals, literals, and punctuation to obtain expressions) and its semantics (what the program does when we run it). Once morphology, syntax, and semantics are mastered, we can start programming, and for this, we need to memorize some keywords and some syntax rules. The amount of such rules is relatively small, and we can write down a *complete* list of them, looking into the code of an interpreter or compiler of the given programming language. For example, the morphology and syntax of a programming language as crucial as C can be entirely described by 187 "lexical rules" (keywords, identifiers, punctuation) and 77 "syntax rules" (of the kind: "an assignment expression consists of a variable name followed by an assignment operator followed by an expression") [8]. Break one of these rules, and your program will not compile; respect the rules, and your program will necessarily compile.

In the case of natural languages, it is virtually impossible to write down *all* rules of a given language, as Henrique [8] has done for C. Not only would there be too many of them (thousands? tens of thousands?), but we wouldn't even agree on many of them since each one of us speaks our own version of a given language (an *idiolect*). While in programming languages, there is a crisp decision (taken by the compiler or interpreter of the language) on the fact that a document (= a program) is morphosyntactically *correct* or not, in natural language, there is a continuum of more-or-less *acceptable* forms, defined by Chomsky as

> [forms] that are perfectly natural and immediately comprehensible without paper-and-pencil analysis and in no way bizarre or outlandish. [3, p. 10]

Acceptability in the use of language has been studied, and formal models have been introduced to deal with it [12]. Still, it remains an important hurdle for computational approaches to language. It also raises another important issue about language, namely the attitude one has towards it. It can be *descriptive* (when we describe it, an attitude that linguists have or claim to have) or *prescriptive* (when we assert rules to be followed by language users, as in language reforms or grammar manuals), or even *proscriptive* (when we limit ourselves to stating prohibitions). But however prescriptive or proscriptive the various authorities may be, language always evolves, and the acceptability of linguistic productions is under permanent fluctuation. In the following (humoristic) example, we can witness the Doctor attempting a new form, facing the (proscriptive) reactions of Clara and concluding by the (pseudo-)linguistic argument that "a group of words including a verb is necessarily a sentence":

> DOCTOR: We're going always.
> CLARA: 'We're going always.'
> DOCTOR: Totally.
> CLARA: That's not actually a sentence.
> DOCTOR: Well it's got a verb in it. *Hide* S33E9

How do we deal with a phenomenon as complex and evolving as language? In 1970, the ground-breaking linguist Richard Montague wrote a paper with the quite ☞ 167 provocative title: "English as a formal language" [18], advocating the use of First-Order Logic methods to describe any natural language, and English in particular. In the second paragraph of this text, he admits that he restricts himself to a "very limited fragment" of English, contradicting the generality of his title. Nevertheless, this work is seminal because it has introduced an approach for considering and processing natural language: expect it to be well-structured, process at least the parts that are processable by your system, and ignore the rest.

Another approach, the *statistical approach*, consists of treating language data like we treat natural phenomena: we monitor them, try to find general laws that describe their behavior, and make statistical predictions. We know in advance that our methods will not provide perfect results, but we hope that the more our algorithms learn, the better the results will be.

In real life, when solving a problem is more important than keeping the same approach throughout, one will often use *hybrid* approaches, starting symbolically and then switching to learning. The first step in machine learning vocabulary is "data preprocessing" before applying the heavy artillery of machine learning algorithms. The "pre-" in "preprocessing" may give the impression that it generally is a quick and effortless step but, in fact, one can apply the most sophisticated symbolic methods during that step, making preprocessing the heaviest and most important part of the process.

1.2 Principles of Linguistics and of Language

Now that we have surveyed the many kinds of language and mentioned some difficulties arising from the naturalness of natural languages, we may legitimately ask questions such as: "Where to start?" or "How do I start studying/processing the "amorphous mass" that is language?"

1.2.1 Signifier and Signified

Ferdinand de Saussure, commonly assumed to be one of the "fathers" of linguistics, came up with the following idea: when we communicate in the oral mode, we use sound to transmit meaning, but there is neither a standard way of segmenting the sounds we hear into units nor a standard way to guess the meaning purveyed by the sounds, *unless* we search simultaneously for sound+meaning combinations. He calls such a combination a *sign*. A sign necessarily has two faces: the *signifier* (a sequence of sounds we perceive) and the *signified* (the meaning carried by the sign). Our brain can find signs in this amorphous mass and to create signs (a process called *semiosis*), thus making communication possible.

1.2.2 Opposition, Etics and Emics

A significant trend in linguistics that reached its highest momentum in the 1960s, *structuralism*, went further in saying that signs of a specific language lived in a *system*, the elements of which are defined by *opposition* [4].

Opposition is a very important principle that we will use throughout this book. The idea is that we distinguish the elements of a set by the fact that some operation applied to them gives different results. The prototypical example is the definition of ☞ 23 *phoneme*: take the English words "cat" and "hat," they have different meanings and are pronounced the same way, except for their beginnings (the [k] and [h] sounds). Because of the existence of this "minimal pair of words" (*minimal* because they only differ by a single sound), we can assert that there exist two elementary units in the phonetic system of English corresponding to the two different sounds of those two words, and which we denote by /k/ and /h/ (using slashes instead of brackets). We first build such a set of phonemes using minimal pairs of words, and then we go a step further—we ask ourselves: What are the features that distinguish the phonemes of our set? We describe their differences by properties they share (or not) and obtain a *space of features* where we can place these elements of our system.

Using opposition to move from phon*etics* (the study of phones) to phon*emics* (the study of phonemes) has inspired scientists in other fields. In 1954, the anthropologist Kenneth Pike invented the terms *emics* and *etics* [19, Chap. 2] as general methods of study in the human sciences: *emics* would be the "insider's view," i.e., how do the members of a community consider themselves and each other, and *etics* would be the "outsider's view," i.e., how does a given community compare with other communities around the world.

1.2.3 Paradigmatic Axis and Syntagmatic Axis

Inside a system, like a language system, we have two axes of investigation: the *paradigmatic* axis, where we replace an elementary unit with another elementary unit of the same type, and the *syntagmatic* axis, where we combine elementary units. Again, the prototypical example is one of the parts of speech in a sentence: by studying the behavior of words, we can assign some of them to a class we will call "verbs" and others in a class we will call "nouns." We can move along the paradigmatic axis by replacing a noun in a sentence with another noun. The sentence remains valid, at least on the level of syntax, e.g., if we take sentences following the pattern determiner-noun-verb like in "the boy sleeps," we can replace nouns and verbs and obtain instances such as "the boy sleeps," "the cat eats," "the idea stinks," etc. Moving along the syntagmatic axis involves investigating different kinds of sentences, like noun-verb, noun-verb-determiner-noun, etc. ☞ 66

This approach can be applied to any situation where we have (1) elementary units and (2) a way to combine them. It has been applied, for example, to Russian fairy tales by Propp [20], in which he analyzes events and characters and concludes that there exist 31 *event functions* that involve seven abstract *character functions*. The way to combine the event functions is their temporal order, which is immutable. An astonishing fact of the syntagmatic axis of this system is that event functions may be absent or present in a given fairy tale, but they always appear *in the same order*.

1.2.4 Compositionality

An important principle of language, about which much ink has been spilled, is *compositionality*. According to this principle, *the meaning of a sentence depends on the meanings of its elementary units and on their order*. I.e., to understand the sentence "Human ate shark," it suffices to know the meanings of the words "human," "ate" (the verb "to eat"), and "shark," and to consider that the words are written in this order and not in the equally valid order "shark ate human."

According to the proponents of this principle, without it, we wouldn't be able to understand a sentence we never heard, even though we know the meaning of each of its words. This makes the principle essential for human communication.

But then again, the principle has limitations. First of all, as advocated by the field of pragmatics, the meaning of a sentence also depends on the conditions of its utterance: who said (or wrote) it, when, how, and why. As a typical example, take the sentence "the train leaves in five minutes": it may mean "let us hurry" if the utterer and addressee are in front of the railway station, or "we missed the train" if they are stuck in a traffic jam many miles from the railway station, or it may have it its literal meaning if the utterer is communicating the information to a person unaware of her location, on the phone.

Secondly, there are idioms: when we say that "somebody kicked the bucket," there is usually neither a bucket nor any kicking.

And finally, meaning can sometimes be constructed in different ways. For example, as Johnson [11] notes, if we take the sentence "You can have the dog spoon," the exact words "dog spoon" may mean "the spoon owned by the dog," "the spoon with a dog image on it," or "the spoon commemorating the year of the dog in the Chinese zodiac." Again, only the context of the utterance can allow disambiguation.

1.2.5 Modalities of Language

By modalities of language, we mean, most importantly, the *spoken* modality and the *written* modality.

There are other, less widespread, language modalities such as *braille*, a tactile writing system for visually-impaired people [22]; *whistling*, used in at least eleven regions of the world to convey messages over long distances, sometimes up to 5 km, [23, 17]; and *lip-reading*, a technique used by hearing-impaired people for understanding speech by observing the utterer's lips, [2]). For the latter, a unit has been defined: the *viseme*, a class of distinctive lip, face, and tongue movements obtained by opposition. These three modalities allow (at least in theory) communication of arbitrary linguistic units (morphemes, words, sentences, texts).

A more restricted modality is the one of *air writing*, in which Japanese speakers will kinesthetically draw sinograms on the palm of their hands to disambiguate their oral production [24].

For additional language modalities in speculative fiction, such as telepathy in Alfred Bester's *The Demolished Man*; concurrent utterances from the double speech organ of the Ariekei in China Miéville's *Embassytown*; use of sonar by the Flouwen in Robert L. Forward's *The Flight of the Dragonfly*; or ink-propulsion by the heptapods in Ted Chiang's novel *Story of Your Life*, see Landragin [13] and Haralambous *et al.* [6].

1.2.6 Functions of Language

The Russian linguist Roman Jakobson was a precursor of the French structuralism movement. In [10, pp. 350–377], he defines six functions of language:

1. the *referential* or *denotative* function: language describes the world, e.g., “on June 1st, the weather in Brest is sunny”;
2. the *emotive* or *expressive* function: the sender describes his/her condition, e.g., “I’m exhausted”;
3. the *conative* function: the message can change the behavior of the receiver, e.g., “Be silent!”;
4. the *metalinguistic* function: language about language, e.g., “the word “word” is a noun”;
5. the *phatic* function: the sender tries to establish contact, e.g., “Hello?”;
6. the *poetic* function: the focus is on esthetic form, e.g., “She walks in beauty, like the night // Of cloudless climes and starry skies,” by Byron.

As an example that combines four among the six functions, Hébert [7, pp. 232–240] gives a 1989 Australian poster warning against drinking while driving:

IF YOU DRINK, THEN DRIVE, YOU’RE A BLOODY IDIOT.

This poster has (1) to attract attention (phatic function), (2) to convince (conative function) by appealing to reason (referential function) or emotion (emotive function), and (3) to get people to act (conative function, again). It would be a nice exercise to rephrase this sentence in rimes to invoke the poetic function. It would be much harder to include the metalinguistic function.

Jakobson’s functions of language are a real issue in artificial intelligence: a chatbot needs to determine the function of language used by its human interlocutor to be able to react accordingly.

See p. 190 for a fictive seventh function of language in a French novel.

1.2.7 Sapir-Whorf and the Eskimo Vocabulary Hoax

In 1911, the linguist Boas [1, p. 73], in an introduction to a handbook on the culture of North American Indians, explained that Eskimoes used four different (but morphologically related) terms for “snow on the ground,” “falling snow,” “drifting snow,” and a “snow drift.” This is perfectly reasonable, especially since the last two terms are two ways of referring to the same phenomenon: “drifting snow” is the agent, and “snow drift” is the action. In 1940, Benjamin Lee Whorf, an MIT alumnus (with a degree in chemical engineering) working at the Hartford Fire Insurance Company, wrote an amateur linguistics article in MIT’s promotional magazine *Technological Review*. Here is an excerpt from this paper:

> We have the same word for falling snow, snow on the ground, snow packed hard like ice, slushy snow, wind-driven flying snow- whatever the situation may be. To an Eskimo, this all-inclusive word would be almost unthinkable; he would say that falling snow, slushy snow,

> and so on, are sensuously and operationally different, different things to contend with; he uses different words for them and for other kinds of snow. [25]

This paper became a best-seller of pseudo-linguistics. In it, the four terms have now become seven. Pullum [21, p. 164] relates in a very vivid style, in subsequent references to Whorf's paper, in 1978, the number of terms was fifty (in a play). In 1984, they became one hundred (in an editorial of the *New York Times*) and two hundred (in a TV weather forecast). But the worst is not the miraculous multiplication of the number of terms but the fact that Whorf builds a theory upon the influence of our language's vocabulary on our perception of the world. The now called *Sapir-Whorf Hypothesis* (Sapir was a student of Boas but has never co-authored anything with Whorf) states that language either determines perception (this is the strong version of the hypothesis) or at least influences our perception. This version is called *inguistic relativism*. For example, in the Breton language, there are no distinct terms for "blue" and "green," does this mean that Breton people are less capable of perceptually distinguishing these colors?

The strong version of the hypothesis has influenced many science fiction novels, such as Orwell's ☞ 84 *Nineteen eighty-four* (the totalitarian "Newspeak" version of English), Vance's *The Languages of Pao*, and Delany's ☞ 190 *Babel-17*. Needless to say, theories that link the (quality) of the perception of a people with a non-biological factor, such as language, can be very dangerous. As McHugh [15], concludes

> If I asked you what a film about aliens had in common with the Nazis, your first answer wouldn't be a patently false interpretation of a linguistic theory. But that's the case. Although grounded in a history of influential linguistic anthropology, the Sapir-Whorf Hypothesis is a cautionary tale about oversimplifying the science of how language shapes us. The average person (or author or dictator) often does not care about the intricacies of the empirical hypotheses, and this can have disastrous consequences.

1.3 A Terminological Issue: Data–Information–Knowledge

The terms "data," "information," and "knowledge" are very common, both in everyday language and in specialized knowledge domains. In AI and in this book, they will carry a specific, technical meaning: *data* is a bunch of numeric values of a given *type*: numeric data, image data, sound data, video data, textual data. We can copy, transmit, or store data, but we don't look into it. We don't try to interpret it. Data has no meaning. When data acquires meaning, it becomes *information*. Information is meaningful data. As for *knowledge*, it allows us to detect tendencies in information and make predictions.

For example, let us plug all kinds of sensors into a railway station: cameras, microphones, piezo-electric cells, thermometers, barometers, etc. These sensors transmit *data*. Let us take video data from a camera close to a platform. By recognizing the shape of a train in the image, an algorithm can assert that the train is located at the platform. This is *information*. By looking at the sign on the wagon, an OCR (Optical Character Recognition) algorithm can recognize the train's number, destination, and departure time. This is also information. More powerful algorithms could perform facial recognition and recognize the passengers' identities. This would not be very ethical, but it is still information. A piezo-electric could tell us exactly when a given train leaves. By combining the various types of information, we could assert that train #8630 leaves at 14:07 for Paris from platform 3. This is again information. Collecting this information, we find out that the same train leaves every working day at the same time for the same destination. Therefore, if the next day is a working day, we can predict that a train will leave for the same destination at that time. This is *knowledge*. Maybe the prediction will fail because of a timetable change, a strike, or an accident. But it is a reasonable prediction based on the observations we did. It is the knowledge we have gained.

What is characteristic of these three notions is the difference in data size to represent each. Video data may need megabytes of data, maybe even gigabytes. Information needs only some numbers and character strings, typically a line in a spreadsheet. But several such lines will be needed if we want to *extract* knowledge. Knowledge can be a concise sentence or rule in a formal language: "Every working day, a train leaves at 14:07 for Paris".

A metaphor has been used for data and knowledge: data is sediment, and knowledge is made of gold nuggets. One has to mine data to extract knowledge. This explains the names of disciplines such as "data mining," "knowledge extraction," etc.

To be useable, knowledge has to be represented in some way. For that, we use *knowledge representation* languages, such as logic. But logic is also used to rep- ☞ 275
resent information, which often leads to confusion. When one uses, for example, First-Order Logic formalism to express the fact that Amy is River's mother (the formula would be $\mathsf{mother}(\mathsf{amy}, \mathsf{river})$), this is not knowledge, but information. Knowledge would be a formula representing the fact that "X is the ant of Y, if X is female and if there is Z such that Z is the sibling of X and Z is a parent of Y," represented by the formula $\forall X, Y\ \mathsf{ant}(X, Y) \rightarrow \mathsf{female}(X) \wedge \exists Z\ \mathsf{sibling}(Z, X) \wedge \mathsf{parent}(Z, Y)$. Such a formula can participate in a reasoning mechanism called *inference* and allow conclusions to be drawn on the ant-condition of a given individual. This is knowledge.

To conclude this section, let us mention a use of the term "data," which is not conformant to our definition: a "database" does not contain data, but information (since the meaning of each field in each record is, usually, well-defined and known by the database users).

1.4 Notations

This book introduces some new notations, using typographetic methods better to ☞ 41
separate the various kinds of information in the text.

Examples in languages other than English are included in double quotes, followed by the corresponding language tag, as in "coin"$_{\text{FR}}$ which is not "coin": the former is the French word for “corner,” and the latter is the English word for what a quarter, a dime, a nickel or a penny are.

The language tags used in the book are (in alphabetical order) : `AR` for Arabic, `CZ` for Czech, `DE` for German, `DK` for Danish, `ES` for Spanish, `FI` for Finnish, `FR` for French, `GR` for Greek, `HB` for Hebrew, `ID` for Indonesian, `IE` for Irish, `IT` for Italian, `JP` for Japanese, `LA` for Latin, `LOJ` for Lojban, `RU` for Russian, `SE` for Swedish, `TK` for Turkish, and `ZH` for Chinese.

As these examples are most often followed by their translation, to avoid overloading parentheses, italics, and doubles quotes, we use the delimiters ⌊…⌋, as in “coin”$_{\mathrm{FR}}$ ⌊corner⌋, or “Gift”$_{\mathrm{DE}}$ ⌊poison⌋, or “σοφία”$_{\mathrm{EL}}$ ⌊wisdom⌋.

The reader will sometimes see a small hand with an index pointing to a number
☞ 10 in the margin, with some words underlined, like this. These annotations are internal references to book pages, given by their numbers. We consider that having bibliographical references inside the text, such as “Haralambous [5, p. 620],” is already cumbersome. References to other parts of the book are placed in the margin so that the reader can consider them as prerequisites and look at them before even starting to read the paragraph. The symbol ☞ is called a *manicule* [9] and appeared well before the invention of typography in 12th-century Spanish manuscripts.

1.5 Exercises and Hints

Every chapter contains exercises. These are first set out in the last section of each chapter, called “Exercises.” Hints for all exercises, as well as a wealth of practical, cultural, or simply amusing information, are given in Chapter 15, “Hints and Expected Results for Exercises.” The reader should first read and ponder the exercise statement, and only when she has concocted a plan for solving the exercise (or when she is definitively convinced that she doesn’t know where to start) read the helpful hints.

Supplementary resources are available at the SpringerLink webpage of the book.

1.6 Resources and Errata

Many public domain resources, often adapted by the author, are available to help readers solve the exercises. These are indicated by the symbol 🗋 `foo.txt`. They can be found at the address

https://www.fluxus-editions.fr/a-course-in-nlp/

Finally, pobody’s nerfect, and this books is no exception. Errata can be found at the address

https://www.fluxus-editions.fr/a-course-in-nlp/errata/

Do not hesitate to contribute!

References

[1] Franz Boas. *Handbook of American Indian Languages*. Vol. 1. Washington: Bureau of American Ethnology, 1911.

[2] Ruth Campbell, Barbara Dodd, and Denis K. Burnham. *Hearing by eye II: Advances in the psychology of speechreading and auditory-visual speech*. Psychology Press, 1998.

[3] Noah Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1964.

[4] Marcel Danesi. "Opposition theory and the interconnectedness of language, culture, and cognition". In: *Sign Systems Studies* 37 (2009), pp. 11–42.

[5] Yannis Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O'Reilly, 2007.

[6] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. "Graphemic and Graphetic Methods in Speculative Fiction". In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[7] Louis Hébert. *An Introduction to Applied Semiotics: Tools for Text and Image Analysis*. Routledge, 2019.

[8] Arthur Henrique. "ANSI C grammar, Lex specification". https://www.quut.com/c/ANSI-C-grammar-l.html. 2012.

[9] Keith Houston. *Shady Characters. The Secret Life of Punctuation, Symbols, and Other Typographical Marks*. New York: W. W. Norton, 2013.

[10] Roman Jakobson. "Linguistics and Poetics". In: *Style in Language*. Ed. by T. Sebeok. MIT Press, 1960, pp. 350–377.

[11] Michael Johnson. "Compositionality in Language". Internet Encyclopedia of Philosophy https://iep.utm.edu/compositionality-in-language/.

[12] Frank Keller. "Gradience in Grammar. Experimental and Computational Aspects of Degrees of Grammaticality". PhD thesis. University of Edinburgh, 2000.

[13] Frédéric Landragin. *Comment parler à un alien? Langage et linguistique dans la science-fiction*. Saint Mammès: Le Bélial, 2018.

[14] Peter H. Matthews. *Concise Oxford Dictionary of Linguistics*. 2nd ed. Oxford: Oxford University Press, 2007.

[15] Amanda McHugh. "Mad Scientists, Aliens, and the Nazi Party: What Is the Sapir-Whorf Hypothesis, and Why Should You Care?" The Cambridge Language Collective, https://www.thecambridgelanguagecollective.com/politics-and-society/what-is-the-sapir-whorf-hypothesis-and-why-should-you-care-part-ii. 2022.

[16] *Merriam-Webster's Dictionary of English*. Springfield, MA: Merriam-Webster Inc., 2016.

[17] Julien Meyer. "Description typologique et intelligibilité des langues sifflées, approche linguistique et bioacoustique". PhD thesis. Université Lyon 2, 2005.

[18] Richard Montague. *English as a Formal Language*. Ed. by Bruno Visentini. Edizioni di Communita, 1970, pp. 188–221.

[19] Kenneth L. Pike. *Language in relation to a unified theory of the structure of human behavior.* Glendale, Calif.: Summer Institute of Linguistics, 1954.

[20] Vladimir Propp. *Morphology of the folk tale*. translated by Lawrence Scott. University of Texas Press, 1968.

[21] Geoffrey K. Pullum. *The great Eskimo vocabulary hoax, and other irreverent essays on the study of language*. Chicago, IL: University of Chicago, 1991.

[22] Evelyn J. Rex et al. *Foundations of Braille Literacy*. New York: AFB Press, 1994.

[23] Annie Rialland. "Phonological and Phonetic Aspects of Whistled Languages". In: *Phonology* 22.2 (2005), pp. 237–271.

[24] Margaret Thomas. "Air Writing as a Technique for the Acquisition of Sino-Japanese Characters by Second Language Learners". In: *Language Learning* 65 (2015), pp. 631–659.

[25] Lee Whorf Benjamin. "Science and linguistics". In: *Technology Review (MIT)* 42 (1940), pp. 229–231.

Part I
Linguistics

This part starts with five chapters corresponding to the five main subdisciplines of linguistics: phonetics and phonology (the study of speech), graphetics and graphemics (the study of written language), morphology (the study of elementary units of meaning, i.e., morphemes, but also words, and terms), syntax (the study of the order of words), and semantics/pragmatics (the study of meaning, be it intrinsic or in the context of an utterance). These chapters encompass the ideas, notions, and methods of linguistics of the last few decades. They are followed by a short chapter on a specific kind of artificial languages, called *controlled natural languages*, which are constrained versions of natural languages, allowing processing that is normally only possible with formal languages.

Chapter 2
Phonetics/Phonology

Phonetics studies the sounds humans produce for communication. Phonetics is an etic discipline. Therefore, rather than focusing on a given language, our point of view in this chapter will be that of considering *all* sounds produced by *all* linguistic communities for *all* languages. As in any communication, there are three components:

1. the transmitter, raising the question of how sounds are produced (this corresponds to the subfield of *articulatory phonetics*);
2. the medium: how are they transmitted (*acoustic phonetics*); ☞ 19
3. the receiver: how are they perceived? (*perceptual phonetics*). ☞ 32

2.1 Articulatory Phonetics

The *International Phonetic Alphabet* (IPA), established in the late nineteenth century and constantly revised ever since, is an attempt to classify *phones* (sounds used in human verbal communication) in a three-dimensional grid. First of all, phones are divided into three global *categories*: *pulmonic consonants*, *non-pulmonic consonants* (used mainly in Amerindian and Caucasian languages—we shall not deal with them here), and *vowels*. This is the first dimension of the grid. Next, consonant phones are classified by the *place of articulation* and the *manner of articulation*, while vowel phones are classified by their *backness* and their *height*. There is a fourth dimension for (pulmonic) consonants, namely, *voicedness*. We will explain these dimensions in the following sections.

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_2

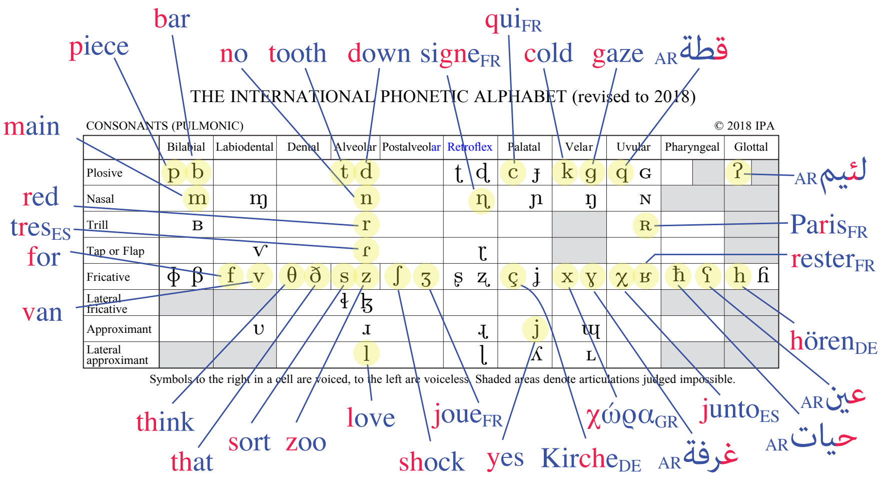


Fig. 2.1 The IPA table of pulmonic consonants (based on the 2018 table on the official Web site of the International Phonetic Association `https://www.internationalphoneticassociation.org/IPAcharts/inter_chart_2018/IPA_2018.html`)

2.1.1 (Pulmonic) Consonants

> Putting aside the sufferings of the early martyrs, few men, I suppose, have gone through more than I myself went through in trying to attain the correct pronunciation of the German word for church—“Kirche.” Long before I had done with it, I had determined never to go to church in Germany, rather than be bothered with it.
>
> Jerome K. Jerome, *Three Men on the Bummel*

The reader can see the IPA table of (pulmonic) consonants in Figure 2.1. We have added English word examples for most of the phones. For phones for which we couldn’t locate any widespread English-language examples, we have chosen words from French, German, Spanish, Greek, and Arabic, as pronounced by native speakers. French words are “signe”$_{\text{FR}}$ ⌊sign⌋, “qui”$_{\text{FR}}$ ⌊who⌋, “Paris”$_{\text{FR}}$ ⌊Paris⌋ and “rester”$_{\text{FR}}$ ⌊to stay⌋, “joue”$_{\text{FR}}$ ⌊cheek⌋. The use of consonant [r] (and consonants close to it) is called *rhoticity*. As for French rhoticity, there are at least four pronunciations of “r”$_{\text{FR}}$ in French. These are: [ʁ] for the standard, “guttural” version, [ʀ] for the “Parisian” variant (called *r grasseyé* in French), [r] for the apical variant (*r roulé*) used mostly by speakers of Southern dialects, and [ɾ] (*r battu*) [26]. German words in Figure 2.1 are “Kirche”$_{\text{DE}}$ ⌊church⌋, as in Jerome K. Jerome’s quote above, and “hören”$_{\text{DE}}$ ⌊to hear⌋. Spanish words are “tres”$_{\text{ES}}$ ⌊three⌋ for [ɾ] and “junto”$_{\text{ES}}$ ⌊together⌋ for [χ], a phone also used in the Scouse pronunciation of “clock” (Scouse is the dialect of Liverpool, whose accent is internationally renowned thanks to the Beatles). The Greek word is “χώρα”$_{\text{EL}}$ ⌊country⌋, the same phone [x] is used in the Scottish word *loch* (place),

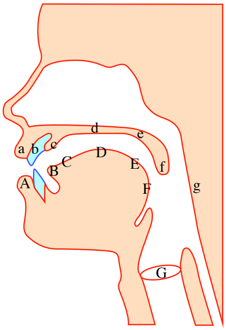


Fig. 2.2 consonants for consonants

as in Loch Ness, where lives Nessie the Monster. As for the four Arabic words, we represent uvular [q] by "قطة"$_{\text{AR}}$ ⌊cat⌋; the glottal stop [ʔ] by "مئيم"$_{\text{AR}}$ ⌊mean⌋, this phone is also used in German between components of a composite word, where the first component ends with a consonant and the second with a vowel, as in "Abart"$_{\text{DE}}$ (pronounced ab-art); the pharyngeal voiced [ʕ] by "عين"$_{\text{AR}}$ ⌊eye⌋, but also the name of the letter itself; the pharyngeal unvoiced [ħ] by "حيات"$_{\text{AR}}$ ⌊life, as in the famous newspaper *Al-Hayat*⌋; and the velar [ɣ] by "غرفة"$_{\text{AR}}$ ⌊room⌋.

Let us see now how these phones are articulated. By *place of articulation* is meant the *narrowing* or *constriction* of two parts of our speech organs during the production of the phone. These parts, called *articulators*, usually go by pair. In Figure 2.2, we have denoted them by letters. The upper articulators are a (upper lip), b (upper teeth), c (alveolar ridge), d (hard palate), e (soft palate), f (uvula), and g (the rear wall of the pharynx). The lower articulators are A (lower lip), B (the tip of the tongue), C (the blade of the tongue), D (the front of the tongue), E (the back of the tongue), F (the root of the tongue) and G (the vocal folds, also called *glottis*).

Using this notation, *bilabial* consonants use the aA articulators, *labiodental* consonants use bA, *dental* consonants use bB, *alveolar* consonants use cB, *postalveolar* consonants use cC, *retroflex* consonants use dB, *palatal* consonants use dD, *velar* consonants use eE, *uvular* consonants use fE, *pharyngeal* consonants use gF and, finally, *glottal* consonants use the glottis G.

As for the *manner of articulation*, it is how phones are produced (rather than the location). One uses the nasal cavity to produce *nasal* consonants. For all other consonants, one uses the oral cavity. To produce *plosive* consonants, the airstream is blocked and released simultaneously with a small explosive noise. The reader can test it by uttering a [p]. *Tap/flap* consonants are like plosives but with a much shorter explosion, so the explosive noise is not heard. *Trill* consonants use a rapid sequence of explosions, with a frequency of around 23 Hz. One encounters such a trill in the Spanish long and powerful [r] of the word "¡arriba!"$_{\text{ES}}$. One leaves only a very narrow opening to produce *fricative* consonants, so the airflow becomes turbulent. This manner of articulation is used by most phones. *Approximants* are like fricatives but with a larger opening, so there is no turbulence. Both fricatives and approximants can be centered or lateral, depending on where the opening is.

The fourth dimension of the description of (pulmonic) consonants is *voicedness*. The reader can try the following test: emit a long [s] sound, turn it into a [z] after a few seconds, and then back to [s] again. By doing this, the reader will notice that the [z] phone opens an air channel that produces a buzzing. This process is called *voicing* and can be applied to plosives and fricative consonants (the vast majority of phones). The same process turns a [t] into a [d], an [f] into a [v], a [θ] (as in "think") into a [ð] (as in "that"), etc. In the table of Figure 2.1, symbols for voiced consonants are represented in the same cell and on the right of their unvoiced versions.

2.1.2 Vowels

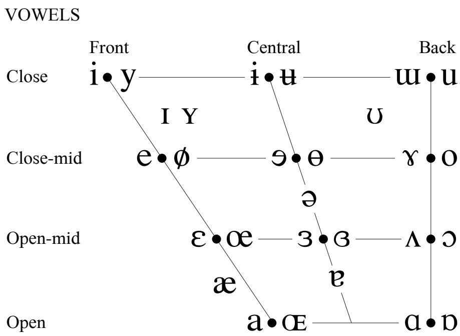


Fig. 2.3 The IPA vowel quadrilateral (taken from https://www.internationalphoneticassociation.org/IPAcharts/inter_chart_2018/IPA_2018.html)

Vowels are produced with a relatively open vocal tract. To classify them, the IPA uses not the metaphor of a table but the one of a trapezium (called *vowel quadrilateral* in [29, p. 10]), represented in Figure 2.3. The reasons for using this strange shape become apparent when we look at the shape and position of the tongue when producing the phones. In Fig. 2.4, we represent the articulations for eight vowels. We have marked on the surface of the tongue the *articulation point*, which is the highest point of the tongue. As we see in Fig. 2.4, this point varies in two dimensions: from front to back (this is called *vowel location*) and from top to bottom (*vowel height*). The vowel quadrilateral uses the same axes: horizontal for location (from front to back of the tongue) and vertical for height ("close" means high and "open" means low). As is evident from Figure 2.4, the more the vowel is low, the less the front-back distinction makes sense. The lower horizontal segment of the vowel quadrilateral is less distinctive and, therefore, shorter, and this accounts for the trapezium metaphor. At the left bottom of the quadrilateral, we find [a], which is, therefore, an open front vowel. This is why the family doctor will ask the patient, 'Say ahh': The tongue is at the front bottom of the oral cavity, and this configuration gives the best view of the larynx.

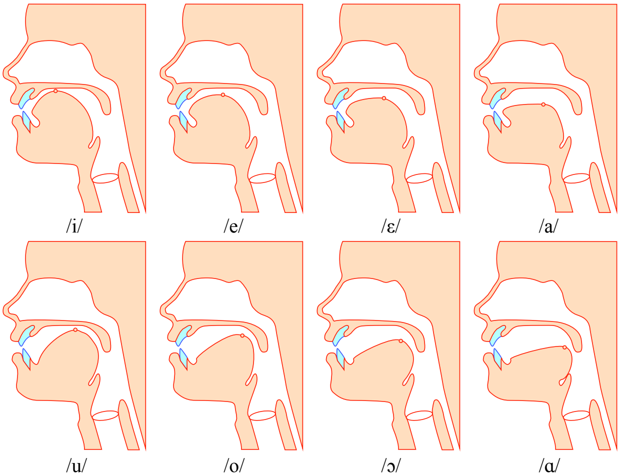


Fig. 2.4 Articulations for eight vowels

A third dimension in vowel classification is the form of the lips, where we distinguish *rounded* and *unrounded* vowels. The symbols for rounded vowels are displayed on the right in the vowel quadrilateral. The reader can test this property by pronouncing the highest and most left vowel, namely the close front unrounded [i] (as in *free*): necessarily, her lips will be spread. By rounding the lips without changing the position of the tongue, she will obtain a close front rounded [y] (as in French “tu”$_{\text{FR}}$ ⌊you⌋ or in German “Glück”$_{\text{DE}}$ ⌊luck⌋).

2.2 Acoustic Phonetics

Acoustic phonetics deals with the transmission of the voice signal. As sound is transmitted through waves, to measure it, we will need its *frequency* (how many waves per second, called *phonation* when we deal with speech) and its *amplitude* (the variation of pressure of the air on the surface of the ear or of the microphone). Besides that, when we have a tube, there is a very important phenomenon (without which there would be no *Flute Sonata in E minor* by Bach, no *Syrinx* by Debussy, and, more generally, no flutes, clarinets, organs, etc.), namely *acoustic resonance*. Thanks to acoustic resonance, every tube (or other volumes) has a *fundamental frequency*. When air transits through the tube, it emits sound in this frequency and in specific higher frequencies called *harmonics*. A flute has holes that allow the player to change the volume and, therefore, the fundamental frequency. Furthermore, the flute’s harmonics make it sound like a flute rather than a rusty bicycle. In the case of our speech organs, we have much more flexibility in changing the volume and shape of the air conduct so that we can modify both the fundamental frequency and the harmonics. In acoustics, these harmonics are called *formants* and written Fn, where F1 is the first formant, F2 the second formant, etc. For $n = 0$, F0 is the fundamental frequency.

It turns out that, at least for vowels, the pair of frequencies of F1 and F2 (in Hz) *unambiguously* characterizes a phone. If we draw F1 and F2 on a plane (with the frequencies in decreasing order on the two axes), we get a shape that looks very much like the vowel quadrilateral of Figure 2.3. To illustrate this, we took the results of a 1995 study [21] on 139 Americans (87% of which were raised in Michigan's lower peninsula and therefore presumably had a standard Inland Northern American English accent) and displayed them on the F1-F2 axis, in Figure 2.5.

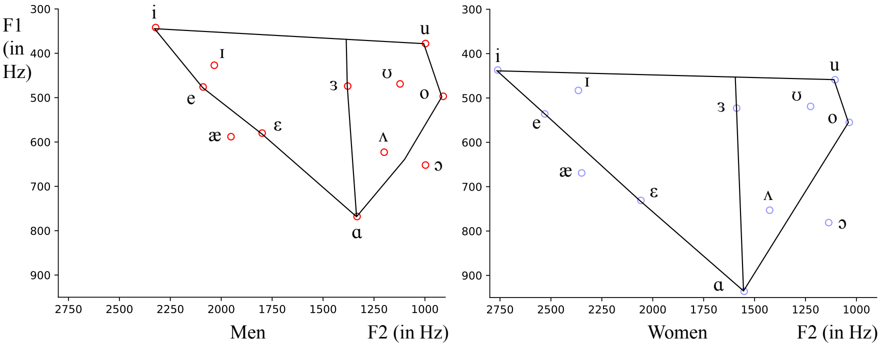


Fig. 2.5 Representation of F1/F2 combinations for various vowels as uttered by native English speakers from Michigan (data taken from [21])

The similarity with the vowel quadrilateral of Figure 2.3 is striking. It suffices to look at the line going from [i] to [ɑ] and passing through [e] and [æ]: it is almost a straight line for men and a perfectly straight line for women.
A very useful tool for studying the physical characteristics of speech is the *spectrogram*. As seen in Figure 2.6, a spectrogram shows all frequencies emitted simultaneously and renders the most intensive ones in darker grey. In Figure 2.6, we see that the F2 value of [i] is highest so that it is located on the left of the triangle of Figure 2.3, while the F1 values of [i] and [u] are practically the same, and the F1 value of [ɑ] is the lowest, and therefore the lower summit of the triangle.

The reader can see a more elaborate spectrogram example in Figure 2.7 where we have gathered four utterances of the iconic sentence "I'm the Doctor" by four Doctors, ranging from 1966 to 2020. Looking carefully, one may observe the higher phonation of the fourth example (the 13th Doctor was a woman), but also the symmetry between the two syllables of the word "Doctor" in the second example (the 4th Doctor) and the emphasis of the first syllable in the third example (by the much more exuberant 11th Doctor).

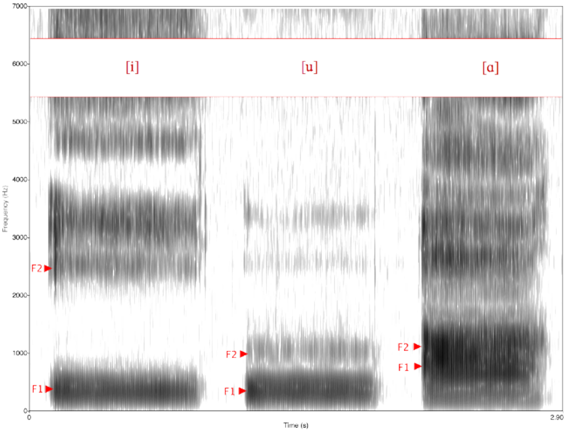


Fig. 2.6 Spectrograms of the American English vowels [i], [uː] and [ɑː] (Wikipedia Commons)

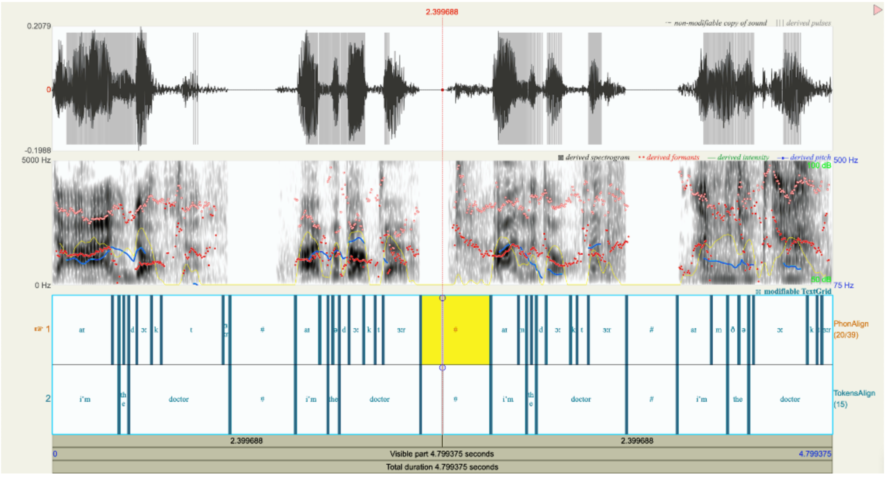


Fig. 2.7 Spectrograms of the sentence “I’m the Doctor” by four Doctors: Patrick Troughton (2nd Doctor, 1966), Tom Baker (4th Doctor, 1978), Matt Smith (11th Doctor, 2011), Jodie Whittaker (13th Doctor, 2020). For phoneme detection and alignment of spectrogram with phonemes, we used SPPAS [3], and for the display, we used Praat [4].

2.3 From Phonetics to Phonemics

2.3.1 Features

The reader has noticed that in the IPA table, various dimensions are used to describe phones. What is interesting is that the dimensions, which we will call *features*, not only describe but actually *define* phones or, more precisely, phone classes. For example, [b] is a voiced bilabial plosive. We can go the other way around and say that we have a continuous spectrum of different realizations of voiced bilabial plosive phones, which we all denote by [b]. Furthermore, we will consider that being voiced, bilabial, and plosive are binary properties so that [b] can be described by a positive value for the three features: [+voiced, +bilabial, +plosive]. Similarly, [p] will be defined by the features [−voiced, +bilabial, +plosive].

In their seminal book *The Sound Pattern of English* [7, pp. 298–329], Chomsky and Halle define some features that do not appear in the IPA table: first of all, to describe vocalic/consonantal characteristics better, they define the non-exclusive features *syllabic* (that can be the peak of a syllable or can form a syllable by itself), *sonorant* (that involves significant opening of the vocal tract), *consonantal* (that involves a major obstruction of the vocal tract). One would expect syllabic phones always to be vowels, but this is not the case: in Czech (or Slovak), phones [l] and [r] can be syllabic, as in the iconic sentence “strč prst skrz krk”$_{\text{cz}}$ ⌊stick your finger through the throat⌋. Vowels are syllabic and sonorant. Some consonants, such as nasals, can be sonorant and consonantal.

Other binary features defined in [7] are *nasal*, *continuant* (whether pronunciation can be prolonged, as in fricatives), *high* (both for vowels and consonants), *low* (for vowels), *back* (for consonants), *coronal* (when the blade or the tip of the tongue are raised), *anterior* (when the place of articulation is located at a frontal position in the mouth), *distributed* (when the constriction extends for a considerable distance along the direction of airflow), *round* (when lips are rounded), etc.

These features are useful because they allow an analytic approach to phone description: one can build binary tables where rows are features and columns are phones, such as in Table 2.1.

Table 2.1 Example of binary table of phone features

	[s]	[z]	[ʃ]	[ʒ]
consonantal	+	+	+	+
continuant	+	+	+	+
voiced	−	+	−	+
anterior	+	+	−	−
coronal	+	+	+	+
distributed	−	−	+	+

☞ 4 Describing phones by binary features is a typically *structuralistic* approach: one first finds elementary features that describe a given object, and one then defines the objects using these elementary features. We will see another application of this approach when we will deal with meaning: Formal Concept Analysis.

2.3.2 Phonemes

Phonemes are classes of phones obtained by *opposition*. They are defined in the frame of a given language so that two phones may belong to the same phoneme in one language but not in another. For example, as we have already mentioned, the alveolar flap [ɾ] and the alveolar trill [r] are *allophones* in French or English (they belong to the same phoneme), but not in Spanish where there is, e.g., a minimal pair [peɾo] (but) vs. [pero] (dog). Orthographically, these two phonemes are distinguished by a single vs. a double <r>: “pero”$_{\text{ES}}$ vs. “perro”$_{\text{ES}}$. Similarly, Greek has no distinction between alveolars and postalveolars, so one can consider that there is a phoneme /s/ (represented by grapheme <σ>), the phone of which is statistically located between alveolar [s], and postalveolars [ʃ].

2.4 Phonological Rules

The major innovation of Chomsky & Halle [7] was the introducing of *phonological rules*. These are formal grammar rewrite rules using a specific notation. A rule, in ☞ 247
this notation [30, pp. 67–71], is of the form

$$\text{focus} \rightarrow \text{structural change} \;/\; \text{left context}\,\_\,\text{right context},$$

where both “focus” and “structural change” are sets of (positive or negative) features, and “context” (left or right) consists of zero, one or more sets of features, followed by an underscore and, again, zero, one or more sets of features. The structural change can be empty, and the contexts can include the symbol #, which denotes the word boundary. The role of the underscore is to distinguish the left context from the right context (which can both be potentially empty).

As an example, let us take the *final-obstruent devoicing* phenomenon in German (*obstruents* are plosive, fricative, or affricate consonants). This will transform a [+voiced] phoneme into a [−voiced] phoneme. For example, the nominative of the word “Rad”$_{\text{DE}}$ ⌊wheel, bicycle⌋ is pronounced /ˈraːt/ while the genitive “Rades”$_{\text{DE}}$ keeps the /d/ phoneme: /ˈraːdəs/ since the /d/ is not final[1]. The corresponding rule will be:

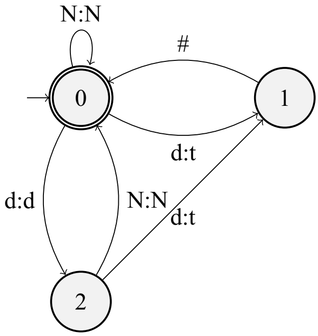
$$[+\text{plosive}, +\text{alveolar}, +\text{voiced}] \rightarrow [-\text{voiced}] \;/\; \_\ \#,$$

in which # (end of word) is the right-side context of the voiced plosive alveolar consonant that, under these circumstances, loses its voicedness. Not writing the +plosive and +alveolar features in the right part means they remain unchanged.

In modern formal grammar notation a rule $A \rightarrow B \;/\; C\ \_\ D$ becomes $CAD \rightarrow CBD$. By observing this rule, one may conclude that the formal grammar to which it belongs is context-sensitive (Type 1 in the Chomsky hierarchy). This is indeed the ☞ 248

[1] The homophony of “Rad”$_{\text{DE}}$ ⌊wheel, bicycle⌋ with “Rat”$_{\text{DE}}$ ⌊advice⌋ has motivated a series of jokes using the proverb “Kommt Zeit, kommt Rat”$_{\text{DE}}$ ⌊time brings advice⌋ that is homophonic with the sentence “Kommt Zeit, kommt Rad”$_{\text{DE}}$: time will bring back the (stolen) bicycle.

case, but as observed in [24] and [18], in most cases, the formal grammars are regular, and phonological rules can be rewritten as non-deterministic finite-state ☞ 261 transducers. As an example, here is the transducer that represents the devoicing of /d/ into /t/ transformation (the symbol N denotes any phoneme besides /d/):



As this transducer is non-deterministic, the replacement of /d/ by /t/ can only occur when it is followed by # (end of the word), which corresponds to a transition to state 1 followed by a transition to state 0; otherwise, one has to go to state 2 (/d/ remains unchanged) and afterward return to state 0.

2.4.1 Underlying Representation

According to [7], the rules presented above are applied not only to existing forms of words but also to abstract forms, called *underlying representations*. Here is an example taken from [34] that attempts to explain the phonological rules of the English past tense. Let us take the three forms: “snowed,” “missed,” and “lifted.” The graphemic representation is similar (a stem followed by <ed>), but we have three different phonemic representations: in the first case, we keep the /d/. In the second, it is devoiced and becomes /t/, and in the third case, a vowel is introduced so that the /d/ is kept. We will use two phonological rules: ɨ-*epenthesis* and *voicing assimilation*:

$$\begin{array}{rl} \text{ɨ-epenthesis} & \emptyset \rightarrow [+\text{syllabic}, +\text{high}, +\text{back}, -\text{round}]\ / \\ & \quad [+\text{coronal}, -\text{continuant}, +\text{anterior}]\ \_ \\ & \quad [+\text{coronal}, -\text{continuant}, +\text{anterior}] \\ \text{voicing assimilation} & [-\text{sonorant}] \rightarrow [-\text{voiced}]\ /\ [-\text{voiced}]\ \_ \end{array}$$

where [+syllabic, +high, +back, –round] corresponds to the phone /ɨ/. The trick is to assume that the underlying representations of the three verbs are /sno/, /mɪs/, and /lɪft/, and the one of the past tense morpheme is /d/. We use a ‘+’ to denote concatenation of morphemes:

Underlying representation	/sno/+/d/	/mɪs/+/d/	/lɪft/+/d/
Applying ɨ-epenthesis	—	—	/lɪftɨd/
Applying voicing assimilation	—	/mɪst/	—
Surface representation	/snod/	/mɪst/	/lɪftɨd/

The example also shows that if the rules are applied in the wrong order, we get the wrong result: /lɪft/ + /d/ becomes /lɪftt/ by voicing assimilation and then /lɪftɨt/ by ɨ-epenthesis, which is wrong.

As an extension of Chomsky & Halle's [7] approach, *Optimality Theory* is based on constraints rather than underlying forms and transformations. The interested reader may find more information in [34, Chapter 14].

2.5 Suprasegmental Aspects

Utterances are not just continuous flows of phones. They also have an inherent structure, which we will briefly describe in this section. It turns out that this structure takes the form of a rooted tree called *prosodic hierarchy*, where the root is the utterance and the leaves are phonemes. In Figure 2.8, the reader can see two examples, taken from [20, pp. 202–203], of different prosodic hierarchies for the same sentence: "On Tuesdays, he gives the Chinese dishes" (uttered in two different ways):

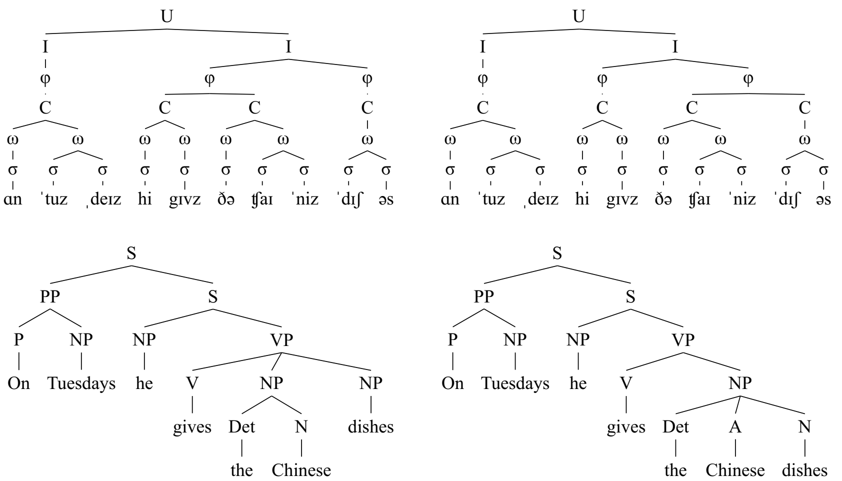


Fig. 2.8 Prosodic hierarchy trees and constituent trees of two interpretations of the sentence "On Tuesdays, he gives the Chinese dishes" [20].

In these trees, U is the utterance. Underneath it, we have five levels of phonetic groups: I (intonational phrases), φ (phonological phrases), C (phonological clitic

Table 2.2 Correspondence between phoneme classes and values of the sonority function, defined in [34, p. 334]

phoneme classes	value of function
low vowels (a, œ, ɑ, ɒ)	8
mid vowels (ɛ, ɜ, ʌ, ə, e, o, …)	7
high vowels and glides (i, y, ɪ, ɨ, u, j, w, …)	6
rhotics (r, ɾ, ʀ, …)	5
laterals (l, …)	4
nasals (m, n, ŋ, …)	3
fricatives (f, v, θ, ð, s, z, ʃ, ʒ, x, ɣ, χ, ʁ, ħ, ʕ, h, …)	2
plosives (p, b, t, d, c, k, g, q, ʔ, …)	1

groups), ω (phonological words), and σ (syllables). In the same figure, we also display the constituent trees corresponding to the two different interpretations of the sentence "On Tuesdays, he gives the Chinese dishes": in the first case, dishes are given to Chinese people, while in the second case, dishes that happen to be Chinese are given to unspecified recipients. Notice that the prosodic and syntactic trees of the two interpretations have similarities but are not identical. ☞ 98

2.5.1 Syllables

As word subdivisions, *syllables* are a concept we all learned in school. They are mostly of the form C*VC* (where C denotes a consonant, V a vowel, and * denotes zero, one or more times, as in regular expression syntax). We call the central part of the syllable (the vowel, if there is one, as in most cases), *nucleus*. What precedes the nucleus is the *onset*, and what follows it is the *coda*. ☞ 249

How do we identify syllables in a word? [34, p. 334] proposes the following method: define a smooth function from phonemes to $\mathbb{R}$ that takes the values displayed on Table 2.2. Syllables will be the phonemes surrounding the local maxima of this function. We call this function the *sonority function* (it is a refinement of the sonorant feature, which is binary).

In Figure 2.9, we have displayed the sonority function values of the sentence "On Tuesdays, he gives the Chinese dishes." Boxes contain the local maxima parts of curves (in the diagram, value 0 of the function denotes word boundaries).

However, this approach bears ambiguities since it is unclear what surrounding phones belong to each local maximum. In our example, there are two such cases: /ʧaɪniz/, which could be split into /ʧaɪ-niz/ or /ʧaɪn-iz/, and /tuzdeɪz/ which can be split into /tu-zdeɪz/, /tuz-deɪz/ or /tuzd-eɪz/. To enable a choice between these alternatives, a principle called the *Maximal Onset Principle* claims that the onset tends to be maximal. According to this principle, /ʧaɪ-niz/, /tu-zdeɪz/ are optimal syllabifications. But /zdeɪz/ doesn't sound right. How can we show it is not a syllable? This is where *phonotactics* steps in. Phonotactics provides us with rules about consonants that can appear in the same part of a syllable. For example, a phonotactic rule states

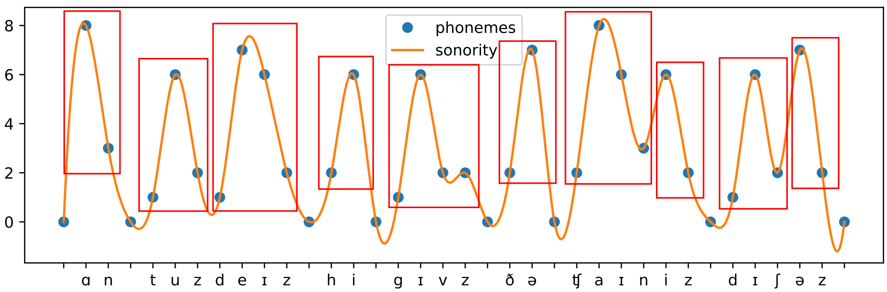


Fig. 2.9 Sonority function of the sentence "On Tuesdays, he gives the Chinese dishes"

that *consonants that share a place of articulation do not occur in the same onset* [17, p. 80]. /z/ and /d/ are both alveolar; therefore /zdeɪz/ cannot be a syllable. The only possible remaining choice is /tuz-deɪz/.

Suppose we take the traditional classification of phones into consonants and vowels (considering that /aɪ/ and /eɪ/ are *diphthongs* and count as a single long vowel), we see that the syllables of our Chinese dishes example are of form CV (/ɑn/, /hi/, /ðə/, /ʧaɪ/), CVC (/tuz/, /deɪz/, /nəz/, /dɪʃ/), and, in a single case, VC (/iz/) and CVCC (/gɪvz/). These counts are conformant to the distribution of English syllable types: according to Dauer [8], 34% of syllables are of type CVC, 30% of type CV, 15% of type VC, 8% of type V, and 6% of type CVCC.

2.5.2 Stress and Foot

One distinguishes between *stressed* and *unstressed syllables* in many languages. Sometimes stress has incidences on morphology (as in English where the word "object" is stressed on the first syllable when it is a noun and on the second when it is a verb), sometimes it has incidences on semantics, as in Greek where we have cases such as "πότε"$_{\text{EL}}$ ⌊where?⌋ and "ποτέ"$_{\text{EL}}$ ⌊never⌋, where the accent denotes stress. Some languages have primary and secondary stress. Typical examples are German compound words such as "**Supp**penre<u>zept</u>"$_{\text{DE}}$, where we have boldened the syllable carrying ☞ 73 the primary stress and underlined the syllable carrying the secondary stress: in fact, compound words have a hierarchic structure, and "Suppe"$_{\text{DE}}$ is the head so that the primary stress of the component becomes the primary stress of the compound word.

Another stress-related phenomenon that occurs in some languages is *vowel reduction* or *lenition* (phones of unstressed syllables become weaker or more open), which is often accompanied by *fortition* (phones of stressed syllables become stronger or more constricted). Consider, for example, the word "atom" /ˈætəm/. When we derive it into the adjective "atomic" /əˈtʰɑmɪk/, the first syllable of which is lenified ☞ 71 (from /æ/ to /ə/) and the second syllable fortified (from /ə/ to /ɑ/).

Syllable lenition is very important in Russian phonology. It is systematic and has two levels, depending on the distance from the stressed syllable. For example, in the word “молоко”$_{\text{RU}}$ ⌊milk⌋ /məɫɐˈko/, the last syllable is stressed and pronounced as /o/, the syllable before it is opened /ɐ/, and the first syllable is merely a shwa /ə/. But when we take the adjective derived from this word, namely “молочный”$_{\text{RU}}$ ⌊milky⌋ /mɐˈɫotɕnɨj/, which is stressed on the second syllable, the stressed syllable is indeed a normal /o/ and the first syllable has turned into an /ɐ/.

Considering syllables of a sentence as units that can be stressed or unstressed, we get the *metrical structure* of the sentence. A *foot* is a sequence of syllables, one of which is stressed. When you divide poetry or song lyrics into feet, they often follow a given pattern. In the simplest case, this pattern is based on disyllabic feet. When the first syllable is systematically stressed, we call the meter of the text a *trochee*, and when it is the second, an *iamb*. Here is an example of two verses in iambic 8/6 syllables, where “—” denotes stressed and “⌣” denotes unstressed syllables:

(1)	⌣	—	⌣	—	⌣	—	⌣	—	|	⌣	—	⌣	—	⌣	—
	əˈ	meɪ	zɪŋ	greɪs	haʊ	swit	ðə	saʊnd	|	ðæt	seɪvd	ə	rɛtʃ	laɪk	mi
	a-	ma-	zing	grace,	how	sweet	the	sound	|	that	saved	a	wretch	like	me

2.5.3 Mora

Moras are *sub*syllabic units. In fact, there are languages with long and short vowels, where the long vowels are (at least in theory) twice as long as the short ones. This was the case in ancient Greek (etas and omegas were twice as long as epsilons and omicrons) and is still the case in Japanese, e.g., in the name of the town *Ōsaka*, the initial ‘Ō’ is twice as long as the other syllables (it is also transcribed “おおさか”$_{\text{JA}}$ in the syllabic script hiragana, containing two identical “お”$_{\text{JA}}$ syllables).

2.5.4 Tone

Tone is the pitch of individual syllables. Many languages use it as a means to distinguish homophonic words. In some of them, called *contour tone languages*, the tone evolves according to a melodic scheme. For example, in the case of Mandarin Chinese, four tones are used: 1 = high; 2 = raising; 3 = lowering, and then raising again; 4 = lowering. The reader can see these tones represented by Praat spectrograms in Figure 2.10.
As an amusement, there exists a special kind of tongue twister, called *tonal tongue twisters*, in which similar syllables but with different tones are alternated. Here is an example, in Chinese, where ā, á, ǎ, à represent the four tones:

(2)	媽	媽	罵	馬	的	麻	嗎?
	māː	ma	mà	mǎ	tɤ	máː	ma?

Mother scolded the horse’s hemp?

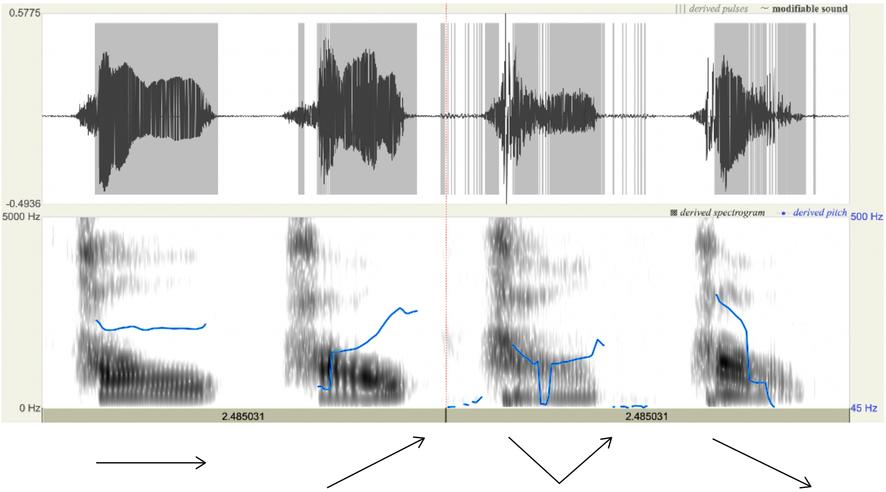


Fig. 2.10 The four tones of Mandarin Chinese, pronounced by Tian Tian. The examples illustrate four times the same syllable *hao*: *hao*^{1} “薅”$_{\mathrm{ZH}}$ ⌊grip⌋, *hao*^{2} “豪”$_{\mathrm{ZH}}$ ⌊generous⌋, *hao*^{3} “好”$_{\mathrm{ZH}}$ ⌊good⌋ and *hao*^{4} “浩”$_{\mathrm{ZH}}$ ⌊immense⌋, represented by Praat spectograms.

2.5.5 Prosody

Prosody deals with the global aspects of spoken utterances. When we analyze music, we use the dimensions of duration/rhythm, melody, and dynamics. In speech, these three dimensions[2] are called *duration* (measured in milliseconds), *energy* (measured in dB) and *intonation* (measured in Hz for the F0 frequency).

Phrase structure is often reflected in prosody, as we have seen in Figure 2.8.

Sometimes, prosody is motivated by the communicative intention of the utterance. For example, we often raise the F0 to the question mark in questions. This is called *question rise*, and the prosodic property is called *tune*. The answer to a question will often be symmetric in terms of tune (lowering the F0), called *final fall*. Tune ambiguity has motivated the famous Jewish “Trotsky” joke, where we have italicized prosodically raised words:

> During a gigantic celebration in Red Square after Trotsky had been exiled, Stalin, on Lenin’s great tomb, excitedly raised his hand to still the acclamations: “Comrades! A most historic event! A cablegram—of congratulations—from Trotsky!” The hordes cheered, and Stalin read the historic cable aloud:
>
> STALIN
> YOU WERE RIGHT, AND I WAS WRONG. YOU ARE THE TRUE HEIR OF LENIN. I SHOULD APOLOGIZE. TROTSKY.
>
> A roar of triumph erupted. But in the front row, a little tailor called, “Pst, Comrade Stalin. A message for the ages! But you didn’t read it with the right feeling!”

---

[2] In music, there is also harmony, as instruments can most pleasantly play simultaneously. In speech, this works out only in very controlled environments such as theater, e.g., in the choruses of Greek tragedies or of contemporary theater [10].

> Whereupon Stalin stilled the throng once more. “Comrades! Here is a simple worker, a loyal communist, who says I haven’t read the message with enough feeling. Come, Comrade, read the historic communication!”
>
> The little tailor went up to the podium, took the telegram, and read:
>
> “Stalin, *You* were right, and *I* was wrong? *You* are the true heir of Lenin?! *I* should apologize?! Trotsky!” [31, pp. 379–380]

The reader may argue that the ambiguity comes simply from the lack of punctuation, but this is actually the key: as [5, p. 17] points out, both the written Hebrew sacred books and this telegram lack punctuation. *Repunctuation* (and, in the oral case, *re-intonation*) is a key tool of midrashic interpretation, and it plays a vital role in the Jewish humor of both the modern and Talmudic periods.

Prosody is very useful in identifying emotions. Table 2.3 correlates the five basic emotions of fear, anger, sadness, happiness, and disgust with six prosodic characteristics: speech rate (rhythm), pitch average (average F0), pitch range (minimum and maximum F0), intensity, voice quality, pitch changes (tune) and articulation.

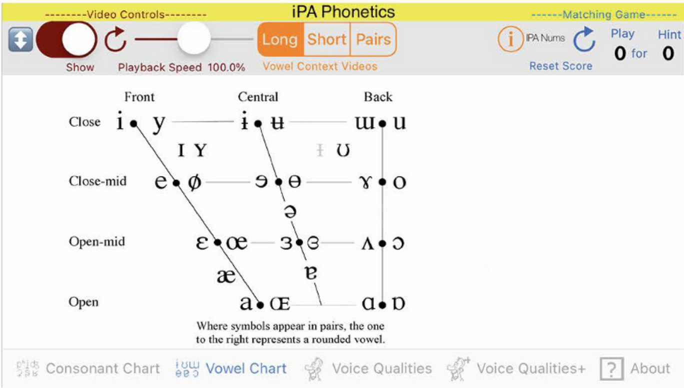
Table 2.3 Correlation of basic emotions and prosodic characteristics [22, p. 27]

	Fear	Anger	Sadness	Happiness	Disgust
Speech rate	much faster	slightly faster	slightly lower	faster or lower	very much slower
Pitch average	very much higher	very much higher	slightly lower	much higher	very much lower
Pitch range	much wider	much wider	slightly narrower	much wider	slightly wider
Intensity	normal	higher	lower	higher	lower
Voice quality	irregular voicing	breathy chest tone	resonant	breathy blaring	grumbled chest tone
Pitch changes	normal	abrupt on stressed syllables	downward inflections	smooth upward inflections	wide downward terminal inflections
Articulation	precise	tense	slurring	normal	normal

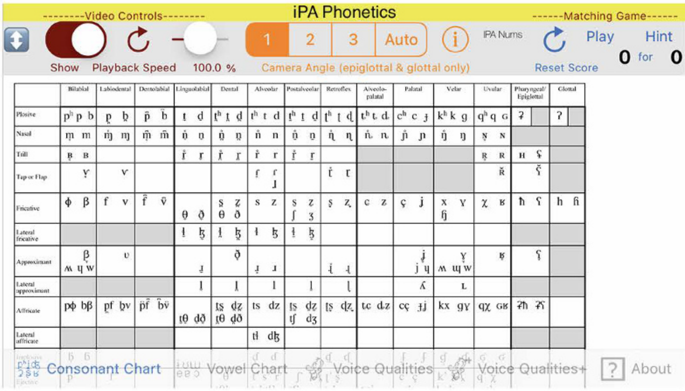
2.6 *iPA Phonetics*, an App for Learning Phonetics

There are many tools for learning the phonology of English, but very few for those aiming at learning phonetics in general. A very interesting tool is *iPA Phonetics*, a free app for iPhone and iPad. According to [13],

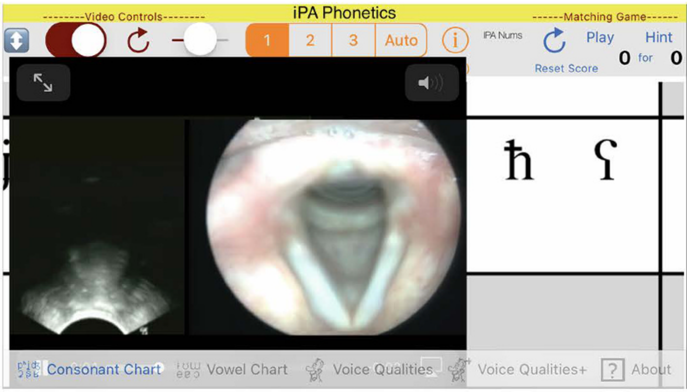
> iPA Phonetics is an iOS application based on the Laryngeal Articulator Model that illustrates voice qualities, vowel qualities, and consonant production with video/audio of articulations in the oral vocal tract and laryngoscopic video/audio and ultrasound images of the laryngeal vocal tract.



(a)



(b)



(c)

Fig. 2.11 Three views of the *iPA Phonetics* application: (a) the consonant chart; (b) the vowel chart; (c) the video and ultrasound views of the lower throat where the articulation/audio of the voiced pharyngeal fricative can be played.

The Laryngeal Articulator Model was introduced by Edmonson and Esling in 2006 [11]. Before this model, there was a consensus in phonetics concerning the airflow through the throat, namely that there is a continuum between the open and the closed glottis (intermediate steps being "voiceless," "breathy," "modal," and "creaky"). The new model is based on six "valves" in the lower throat area, the first valve being the glottal vocal folds per se. In [11], examples for articulating the six valves are given, taken from languages worldwide.

The application displays an IPA consonant chart that is more detailed than the one we gave on p. 16, taken from [12] as well as the standard vowel chart with an additional symbol "ɟ" (both charts are "logical extension[s] and economic reorganization[s] of the symbol categories in the 2005 IPA chart[s]" [13]. In both charts, clicking on an arbitrary phonetic symbol loads resources: videos of the mouth or the lower throat, audio files, and even ultrasound images.

In Fig. 2.11, we display the two charts and a screen capture from the lower throat video and ultrasound image corresponding to the voiced pharyngeal fricative. This phone represents the phoneme /ʕ/ (grapheme "ع"$_{AR}$) of the Arabic language, as in the word "عين"$_{AR}$ *eye*, one of the most difficult phonemes to master for learners of Arabic as a second language and one of the most representative of this language's phonology.

Well understood, there is no miracle solution, and by hearing the audio files and watching the video files, the user will not instantly master the phones of all languages. Still, it is the closest one can get to this mastery without the help of an instructor in phonetics or of a native speaker of each language.

2.7 Psycholinguistic Aspects, Perceptual Phonetics

In this section (heavily inspired by [32]), we will deal with the psychological aspects of auditory language perception. Our ears are most sensitive to frequency ranges used by speech: between 600 Hz and 4,000 Hz (to compare, whales communicate in the range of 30 Hz to 8,000 Hz). Furthermore, our brains are splitting language and non-language sound streams and processing them differently. In an experiment [33], some phonemes have been replaced by non-speech sounds such as coughs. This happened because listeners have barely noticed them (this is called *phoneme restoration effect*). They kept the most plausible interpretation when there was ambiguity without even noticing that the phoneme was missing.

There is enormous variability in auditory input depending on who is speaking and under what circumstances. Therefore, it has been argued that our memory system stores *multiple representations* for a given unit. By unit, we mean a phoneme, word, or other linguistic unit. These representations are called *exemplars*, and we store quite a few of them, together with information on the speaker's age, sex, social group, the time and place of utterance, etc. This allows us to recognize the same unit in different contexts and anticipate its occurrence in a given context.

An important perception factor is *signal continuity*. Our brains have been prepared by evolution for millennia so that nowadays, we can follow a distinct speaker amid the noise of a cocktail party. This is the *Cocktail Party Effect*.

As for identifying phonemes, listeners gather multiple cues and integrate them to identify a phoneme correctly. This also heavily involves *lip reading*, whenever possible. The importance of the influence of visual cues on our perception of phonemes has been shown through a spectacular experiment called the *McGurk Effect* [28]. In this experiment,[3] one first hears and correctly sees a person uttering several times the syllable /ba/. Then, while keeping the same soundtrack, the video shows the same person uttering a different phoneme. As a result, the subject perceives a different phoneme. Many psychological illusions become ineffective once explained; this is not the case with the McGurk Effect, which remains valid whenever the video is watched. Besides vision, stimuli on our skin can also interfere with auditory perception: it has been observed [16] that because of a puff of air directed at the hand or the neck, an (unaspirated) /b/ is perceived as an (aspirated) /p/.

Identifying phonemes is a classification process—this becomes clear through an experiment where subjects hear a continuum of phones between two phoneme prototypes, for example, between /b/ and /p/. It turns out that there is a specific boundary where the perception of the phoneme changes from /b/ to /p/. And it seems this is reported even for 3-month-old infants, long before they start speaking. In case the reader is wondering, the auditory perceptual abilities of infants are measured using a procedure classed *high amplitude sucking* (HAS) [14, p. 177]:

> In the classic HAS paradigm, an auditory stimulus is presented contingent on the infant's high-amplitude sucks: each time the infant produces a suck above a predefined amplitude threshold, the infant receives an auditory stimulus. [...] Once a preestablished satiation criterion has been reached (a decrease in the sucking rates), the stimulus is changed for the subjects in the Experimental group. The change of stimulus is known as the "shift." To monitor for spontaneous changes in infants' sucking, a Control group undergoes similar conditions as the Experimental group, but the stimulus remains unchanged after the satiation criterion has been reached. A significant difference in the post-shift sucking rate between the Control group and the Experimental group is interpreted as evidence that the two stimuli were discriminated by the infants.

To return to non-infants, the boundary between phonetic categories also depends on linguistic information (this is called the *Ganong Effect* [15]): in the /d/-/t/ continuum, for example, the boundary is nearer to the /d/ end for the stimulus /?ask/ (as in *task*) and nearer to the /t/ end for the stimulus /?esk/ (as in *desk*).

[3] See, e.g., `https://www.youtube.com/watch?v=2k8fHR9jKVM`.

2.8 Further Reading

2.8.1 Literature

The best introductory books to Phonetics and Phonology are, respectively, Ashby & Maidment [2] and Odden [30], from the Cambridge series *Cambridge Introductions to Language and Linguistics*. The books in this collection are easily recognizable by their "Introducing…" titles.

The corresponding Oxford series "Oxford Textbooks in Linguistics" has an equally well-written volume on Phonetics by Catford [6]. This book is more "practical" in the sense that it concentrates on methods to distinguish phones, whether heard or produced. It contains 124 exercises (without solutions) and a "Further Reading" section. Ashby & Maidment [2] is more general and has an extensive iconography (spectrograms, medical images, etc.). It contains 70 exercises (with solutions, and this can be pretty useful for self-learners) and a glossary.

As for phonology, Odden [30] gives an elaborate and easy-to-follow introduction to underlying representations, using the notation we introduced in § 2.4.

At a more advanced level, de Lacy [25] is a handbook containing 25 chapters on phonology written by different authors.

For a thorough introduction to the acoustic analysis of speech, we recommend Martin [27]. It covers the physics of sound and Fast Fourier Transforms, wavelets, articulatory models, spectrograms, etc.

For those interested in the history of Phonology, a complete handbook has been published very recently: Dresher & Van der Hulst [9]. Its 880 pages contain 33 chapters ranging from antiquity to the 21st century and beyond. Concerning the history of the International Phonetic Alphabet, one may consult Albright [1].

2.8.2 LaTeX

The *tipa* package[4] allows typesetting of IPA symbols using macros. When using a Unicode-compliant version of the TeX engine (such as XeTeX) and Unicode fonts (such as *FreeSerif*), IPA symbols can also be inserted directly into a document, the advantage of *tipa* is that it allows arbitrary diacritics to be applied to symbols.

To draw vowel quadrilaterals, one can use the *vowel* package.[5]

To typeset glosses (multiples lines of text and annotations, horizontally aligned) as on pages 28 and 28, one can use the *expex* package,[6]. It has a very user-friendly syntax.

For drawing syntax trees and prosodic trees, see p. 130.

[4] https://ctan.org/pkg/tipa

[5] https://ctan.org/pkg/vowel

[6] https://ctan.org/pkg/expex

2.8.3 Science Fiction

A recurring criticism of science fiction is that the aliens miraculously seem all English-speaking [23, p. 364]. This criticism already commits a sin of anthropomorphism since it takes the fact that they communicate orally for granted. Some authors have proposed more imaginative visions of alien communication that push the boundaries of phonology.

In his novel *Embassytown* (2011), Miéville describes a civilization of aliens, the Ariekei, communicating orally by using not one but two speech organs emitting simultaneously. For the Ariekei, human single-channel oral communication is not language. For them, utterances must be two-fold, emitted by two synchronized sources. By the simultaneity of semantically different utterances, they achieve various communicative intentions such as doubt (by simultaneously uttering a question and an assertion) [19, pp. 310–312]. To communicate with them, humans have genetically engineered monozygotic twins sharing the same mental processes and therefore speaking simultaneously [19, p. 283].

Often, the communication medium is different from the human voice. This medium may be sound but in much higher frequencies than what is usual for humans, like in the S3E11 episode of *Star Trek*, "Wink of an Eye" (1968), where the Scalosians live in a different time scale—or in much lower frequencies, like the Selenites in Kuttner's *The big night* (1947). In his novel *The Flight of the Dragonfly* (1984), Forward describes a very intelligent but non-technological species, the Flouwen, communicating with human visitors on their planet through a sonar.

In some cases, aliens communicate not through sound but through electromagnetic radiation, as in Leinster's novelette *First Contact* (1945), using microwaves emitted from an organ in their heads. Sound waves are undetectable to them, as are microwaves to humans, and therefore, according to Leinster, "From [the humans'] point of view, they have telepathy. Of course, from their point of view, so do we."

2.9 Exercises

Exercise 1-1: English Accents

Find

(a) the ten most frequent words and
(b) the ten longest words

that are *pronounced exactly the same in the UK (North), in Scotland, and in the US.*

And as the answer to (a) may be a list of very short words and the one to (b) a list of rare and long words, let us make a compromise and also find

(c) the ten most frequent words comprising six phonemes.

Find the ten words pronounced the most differently in the UK (North) and the US.

Exercise 1-2: Phonotactics of English

Explore the phonotactics of English. Rank the following Whovian named entities in decreasing order of conformance to English phonotactics:

Lethbridge, Zygon, Dalek, Angstrom, Graham, Yrcanos, Yaz, Pting, Zaakros, Susan.

Exercise 1-3: Tonotactics of Vietnamese

Vietnamese is a tone language: six tones are used on a lexical level. Each word has a unique tone, which is used on every occurrence of the word. Explore the *tonotactics* of Vietnamese: the behavior of the tones of sequences of words. In particular, give the distribution of tones and answer the question: if X is the variable that represents the tone of a word, and Y the variable that represents the tone of the preceding word, are X and Y independent? I.e., do we have $P(X \mid Y) = P(X)$?

Exercise 1-4: Classification of Voice Files

Download files `voice01.wav` to `voice10.wav`. They contain short French phrases uttered by speakers D (five files) and speaker Y (five files). Four of them are interrogative, and six are declarative. Classify them algorithmically (without listening to them) according to the speaker and to interrogative/declarative mode.

References

[1] Robert W. Albright. *The International Phonetic Alphabet: Its Backgrounds and Development*. Indiana University Research Center in Anthropology, Folklore, and Linguistics, 1958.

[2] Michael Ashby and John Maidment. *Introducing Phonetic Science*. Cambridge University Press, 2005.

[3] Brigitte Bigi. “SPPAS, Multi-lingual Approaches to the Automatic Annotation of Speech”. In: *The Phonetician* 111–112 (2015), pp. 54–69.

[4] Paul Boersma and Vincent van Heuven. “Speak and unSpeak with PRAAT”. In: *Glot International* 5 (2001), pp. 341–347.

[5] David Brodsky. “Why Did the Widow Have a Goat in Her Bed? Jewish Humor and Its Roots in the Talmud and Midrash”. In: *Jews and Humor*. Vol. 22. Studies in Jewish Civilization. West Lafayette, IN: Purdue University Press, 2011, pp. 13–32.

[6] John C. Catford. *A Practical Introduction to Phonetics*. 2nd ed. Oxford University Press, 2001.

[7] Noam Chomsky and Morris Halle. *The Sound Pattern of English*. Harper & Row, 1968.

[8] Ryan M. Dauer. “Stress-timing and syllable-timing reanalysed”. In: *Journal of Phonetics* 11 (1983), pp. 51–62.

[9] B. Elan Dresher and Harry van der Hulst. *The Oxford History of Phonology*. Oxford University Press, 2022.

[10] Helen Eastman. "Chorus in Contemporary British Theatre". In: *Choruses, Ancient and Modern*. Ed. by Joshua Billings, Felix Budelmann, and Fiona Macintosh. Oxford: Oxford University Press, 2013, pp. 363–376.

[11] Jerold A. Edmondson and John H. Esling. "The valves of the throat and their functioning in tone, vocal register, and stress: Laryngoscopic case studies". In: *Phonology* 23 (2006), pp. 157–191.

[12] John H. Esling. "Phonetic notation". In: *The Handbook of Phonetic Sciences*. Ed. by W. J. Hardcastle, J. Laver, and F. E. Gibbon. Oxford: Wiley-Blackwell, 2010, pp. 678–702.

[13] John H. Esling, Scott R. Moisik, and Christopher Coey. "iPA Phonetics: Multimodal iOS application for phonetics instruction and practice". In: *International Congress of Phonetic Sciences (ICPhS), Glasgow*. 2015.

[14] Caroline Floccia, Anne Christophe, and Josiane Bertoncini. "High-Amplitude Sucking and Newborns: The Quest for Underlying Mechanisms". In: *Journal of Experimental Child Psychology* 64 (1997).

[15] William F. Ganong. "Phonetic categorization in auditory word perception". In: *Journal of Experimental Psychology, Human Perception and Performance* 6 (1980), pp. 110–125.

[16] Bryan Gick and Donald Derrick. "Aero-tactile integration in speech perception". In: *Nature* 462 (2009), pp. 502–504.

[17] Ulrike Gut. *Introduction to English Phonetics and Phonology*. Peter Lang, 2009.

[18] "K + K = 120". In: *Papers dedicated to László Kálmán and András Kornai on the occasion of their 60th birthdays*. Ed. by Beáta Gyuris, Katalin Mády, and Gábor Recski. Budapest: MTA Research Institute for Linguistics, 2019.

[19] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. "Graphemic and Graphetic Methods in Speculative Fiction". In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[20] Bruce Hayes. "The prosodic hierarchy in meter". In: *Rhythm and meter* (1989), pp. 201–260.

[21] James Hillenbrand et al. "Acoustic characteristics of American English vowels". In: *The Journal of the Acoustical Society of America* 97 (1995).

[22] Krzysztof Izdebski. *Emotions in the Human Voice. Volume 3: Culture and Perception*. Plural Publishing Inc., 2007.

[23] Gwyneth Jones. "Aliens in the fourth dimension". In: ed. by Rob Latham. Bloomsbury, 2017, pp. 364–374.

[24] Ronald M. Kaplan and Martin Kay. "Regular models of phonological rule systems". In: *Computational Linguistics* 20 (1994), pp. 331–378.

[25] Paul de Lacy, ed. *The Cambridge Handbook of Phonology*. Cambridge University Press, 2007.

[26] Pierre Léon. "Le classement et le fonctionnement des consonnes". In: *Phonétisme et prononciations du français*. Ed. by Pierre Léon. Armand Colin, 2011, pp. 91–110.

[27] Philippe Martin. *Speech Acoustic Analysis*. Wiley & Sons, 2021.

[28] Harry McGurk and John MacDonald. In: *Nature* 264 (1976), pp. 746–748.

[29] Francis Nolan et al. *Handbook of the International Phonetic Association. A Guide to the Use of the International Phonetic Alphabet*. Cambridge: Cambridge University Press, 2003.

[30] David Odden. *Introducing Phonology*. 2nd ed. Cambridge University Press, 2013.

[31] Andrei Rogachevskii and Kenneth W. Wilson. "Lev Trotsky in Anecdotes, Jokes and "Chastushkas" ". In: *New Zealand Slavonic Journal* (1999), pp. 375–384.

[32] Paul Warren. *Introducing Psycholinguistics*. Cambridge University Press, 2013.

[33] Richard M. Warren. "Perceptual Restoration of Missing Speech Sounds". In: *Science* 167 (1970), pp. 392–393.

[34] Elizabeth C. Zsiga. *The Sounds of Language. An Introduction to Phonetics and Phonology*. Wiley-Blackwell, 2013.

Chapter 3
Graphetics/Graphemics

In this chapter, we will deal with the written modality of language. We adopt the same (structuralistic) emic/etic approach as for the oral modality of language. We will first consider the etic approach, *graphetics*, and a special case: *typographetics*. Then we will move to the emic approach, *graphemics*, as well as a particular case, *sinographemics*, the graphemics of the Chinese writing system, used for Chinese and Japanese language (as well as Korean and historical Vietnamese).

3.1 Graphetics

According to Althaus,

> The objects of graphetics as a subdiscipline of linguistics are the conditions and material elements that constitute visual linguistic communication. These are, in particular, the different ways of putting something into writing: scripts, typefaces, variants of writing something, individual and social differentiations, historical developments, and calligraphic and typographic norms. [4] quoted in [41, p. 57]

As in the case of phonetics, we are dealing with communication, so there are three components: the transmitter, raising the question of how written material is produced (this corresponds to the subfield of *productional graphetics*); the medium: the study of the written material per se (*descriptive graphetics*); the receiver: how are they perceived? (*perceptual graphetics*).

3.1.1 Descriptive Graphetics

Graphetics describes written language at the etic level. Meletis & Dürscheid [41, p. 63] define the basic unit of graphetics as being the *graph*. This term can be used unambiguously in a document on (theoretical) linguistics. In this book, as we also introduce the mathematical structure of graph (a set of binary relations represented by ☞ 207

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_3

edges and vertices), to avoid ambiguity, we will denote by γ-graph (γ from "γραφή"$_{\text{EL}}$ ⌊writing⌋) the linguistic notion of graph.

We saw in the previous chapter that phones are described by features (articulation, manner, etc.). How about γ-graphs? There are two main differences between phones and γ-graphs: the former are produced by human speech organs (or voice synthesis algorithms that emulate human speech organs) while the latter mostly need a tool to be produced (calamus, pencil, typewriter, computer, …) and therefore strongly depend on the tool's graphic properties and the creative freedom of the human user of the tool. While phones are linearly ordered by their time of utterance, γ-graphs evolve in two (or even three, see Warmann [60, 'Signage']) spatial dimensions and therefore have more flexibility. Does it still make sense to break down γ-graphs into "features"? This has been attempted for specific scripts, e.g., the Roman script, for which Watt [62] has defined a *kinemic grammar* (movements of the hand) and a *phanemic grammar* (perceptual elements) while Althaus [4] has defined *graphically distinctive features* and *writing areas* (see also Scharnhorst [50] for the Cyrillic script).

A decomposition of graphs can seem dubious since, in a normal reading process, we read entire words and pay little attention to distinctive features of γ-graphs. And word shapes are so redundant in their (word-)distinctive features that for a text to be understood, the upper parts of γ-graphs are sufficient [46], as illustrated by the words the reader is reading right now.

Also, decompositions of γ-graphs have been used in applications such as Optical Character Recognition (OCR). Casey [9] provides an interesting survey of γ-graph segmentation approaches used ever since the 60s. These approaches resulted not only in OCR methods but also in the creation of γ-graphs optimized for the OCR task, such as the ones of the typeface OCR-A, developed in 1966 and containing γ-graphs for three additional graphemes: "chair" ⑁, "fork" ⑂ and "hook" ⑀ [31]. The OCR-A typeface is mainly used for bank checks (and in graphic design connoting technology).

3.1.1.1 Cheirographetics, or the Study of Handwriting

According to a 2021 CBS News poll,[1] most Americans haven't written a personal letter on paper in over five years. Furthermore, rare are the handwritten text corpora[2] and these are created mostly for testing OCR software. Therefore we are going to keep this section short, providing merely some references: A description of handwriting in the 20th century can be found in [49]; analysis of handwriting is useful in two cases: the medical case (e.g., to diagnose neurological disorders [54]) and the forensic case (e.g., to prove that a given individual has written a given piece of evidence [20]); and, finally, the reader should be aware of the fact that *graphology*, i.e., the practice of supposedly determining personality traits of a writer through their written production, is no more than a (quite lucrative) pseudo-science [6, 16].

---

[1] https://perma.cc/M38Q-QQ3D.

[2] See, e.g., https://paperswithcode.com/datasets?task=handwriting-recognition.

3.1.1.2 Typographetics

We call *typographetics* the subfield of graphetics dealing with γ-graphs produced by hot-lead printing machines or by other tools perpetuating the visual structure of hot-lead typography [59], such as the typewriter [2, 58], Monotype [30] and Linotype [48] machines, phototypesetting [39] and, most important nowadays, the computer.

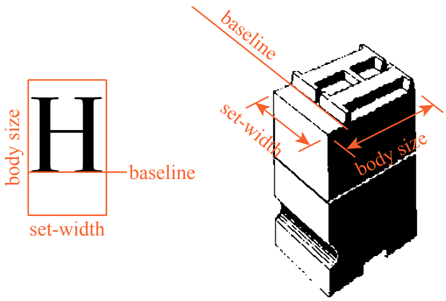


Fig. 3.1 On the left: abstract characteristics of a τ-graph, on the right: their historical origin in the metal type context. Figure taken from [24, p. 10]

A typographetic γ-graph, which we will call *τ-graph* for simplicity (τ from "τυπογραφία"$_{\text{EL}}$, typography), besides being a shape, has two additional features, inherited from metal types: (a) it is contained in an abstract box with a horizontal dimension called *set-width* and a vertical dimension called *body size*, and (b) it includes an abstract horizontal (or vertical, in the case of vertical scripts such as Chinese or Mongolian) line called *baseline* (see Figure 3.1). To these, we can add some additional abstract features such as *uppercase height* and *x-height* for *bicameral scripts* only, that is, scripts featuring uppercase and lowercase letters, namely Roman, Greek, Cyrillic, and Armenian; in the case of Hebrew, which is a *unicameral script*, the specific typeface of the example uses a *Hebrew height* at 40% between x-height and uppercase height (see Figure 3.2).

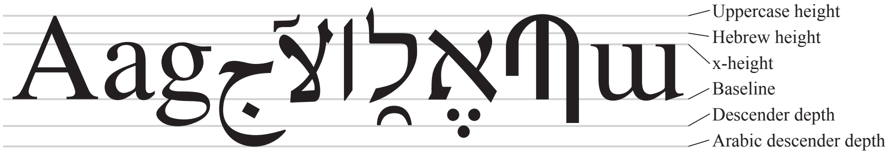



Fig. 3.2 More abstract characteristics of τ-graphs, in a multi-script context: Roman, Arabic, Hebrew and Armenianτ-graphs from a *Times*-like typeface. Figure taken from [24, p. 662]

3.1.1.3 Descriptive Levels

According to Meletis [40, p. 116] (see also [29, pp. 262–268]), there are five levels of study in graphetics, listed below. For each level, we have an elementary unit and a separator of elementary units:

1. the *micrographetic level*, where we study individual τ-graphs; the separator is *intercharacter space*;
2. the *mesographetic level*, where we study *graphetic words*, i.e., groups of τ-graphs; they are separated by *interword space*;
3. the *macrographetic linear level*, where we study *lines*, i.e., sequences of τ-graphs and interword spaces that do not exceed a limit, called *text width*; the separator is called *leading*;
4. the *macrographetic planar level*, where we study *text blocks*, i.e., sequences of graphetic lines (paragraphs, lists, etc.); the separator is *inter-paragraph space*;
5. the *paragraphetic level*, where we study entire *pages*.

To this, inspired by Genette [18], we add another level: the *perigraphetic level*, which deals with the publisher's peritext: titles, dedications, epigraphs, prefaces, intertitles, etc.

Depending on the script, the direction of sequences of elementary units (separated by the corresponding separators) may differ:

Script	Micro	Meso	Macro linear	Macro planar	Para
Roman, Greek, etc.	→	→	↓	↓	→
Arabic, Hebrew, etc.	←	←	↓	↓	←
Vertical Chinese, Japanese	↓	∅	←	←	←
Mongolian	↓	↓	→	→	→

It is interesting to notice that these levels are not always embedded: a graphetic line contains mainly graphetic words, but sometimes these words are broken (hyphenated); a page mainly contains text blocks, but sometimes they are broken between pages.

Let us consider two interesting phenomena at the micrographetic level: *kerning* and *ligatures*, as well as an interesting phenomenon at the mesographetic level: *hyphenation*.

3.1.1.4 Kerning and Ligatures

Kerning is the fine-tuning of intercharacter space to strengthen the visual coherence of graphetic words: a badly kerned text makes the perception of graphetic words more difficult and, therefore, hinders the reading process.

Ligatures are cases where kerning is impossible because of the τ-graphs' shapes; therefore, a more radical technique is used: τ-graphs are merged, as in 'ﬁ' or 'ﬂ' (compare with 'f i' and 'f l').

The reader may wonder why these two purely visual techniques may be interesting from a linguistic point of view. In both cases, the presence or absence of these features can bear linguistic information. Compare the following two sequences of τ-graphs: '*A VA*' and '*AVA*'. In the first case, the letters are kept at a distance, as they should not be read as a graphetic word; they are, in fact, symbols of mathematical variables. In the second case, the sequence is to be read as a graphetic word (a detergent brand or the given name of the great actress Ava Gardner).

As for ligatures, there is a rule in German graphotactics (*graphotactics* is the behavior of τ-graphs in a sequence) stating that ligatures are always *intramorphemic*: compare the words "Auflage"$_{\text{DE}}$ and "Querflöte"$_{\text{DE}}$; in the former, we have two free morphemes ("auf"$_{\text{DE}}$ and "lage"$_{\text{DE}}$), so that the τ-graphs 'f' and 'l' are separated by the morpheme boundary; in the later, we have, once again, two morphemes ("quer"$_{\text{DE}}$ and "flöte"$_{\text{DE}}$) but this time τ-graphs 'f' and 'l' belong to the same morpheme, so that the ligature is allowed.

3.1.1.5 Typographetic Functions and Connotations

To compete with prosodic features of the oral modality, typography has introduced a large number of variables that carry connotative meaning (all section references are from the *Chicago Manual of Style* [57]):

- *Italics* (vs. roman) is used for emphasis §7.50, for unfamiliar words and phrases from other languages §7.53 and §11.3, for highlighting key terms, for legal cases §8.82, for names of ships §8.116, for names of species §8.120, for gene names §8.132, for infections §8.145, for titles of books and periodicals §8.168, for titles of plays §8.183 and stage directions §13.46, for titles of movies §8.189, for video games §8.190, for operas §8.194, for music albums §8.197, and for abbreviations, but only if the expanded version would be italicized in the first place §10.7.
- **Bold** (vs. regular) is used mainly for titles.
- SMALL CAPS are used for statement names in mathematics §12.56, for speaker names in plays §13.46—and in French, for centuries [36, p. 159].
- The Roman/Sans Serif distinction is used when two text blocks have different functions, e.g., the recipe ingredients in a cookbook vs. the cooking instructions.
- `Monospaced type` (vs. variable-width type) is used for computer code, URLs, and the such (these typefaces should also clearly distinguish the τ-graphs of the letter 'O' and the digit zero and of the letter 'I,' the letter 'l' and the digit one).
- The distinction normal spacing/l e t t e r s p a c i n g should be avoided. The great font designer Frederic Goudy once said [55]:

  > Any man that'll letter space lower case w i l l s t e a l s h e e p !

  There is an exception, though: in broken-script German, there is no italic font[3] and letter spacing is the standard way of connotating emphasis: "man ſperrt

[3] Despite some very beautiful attempts such as *Deutsche Kursiv* designed by Richard Ludwig in Frankfurt am Main, in 1909 [23, p. 9].

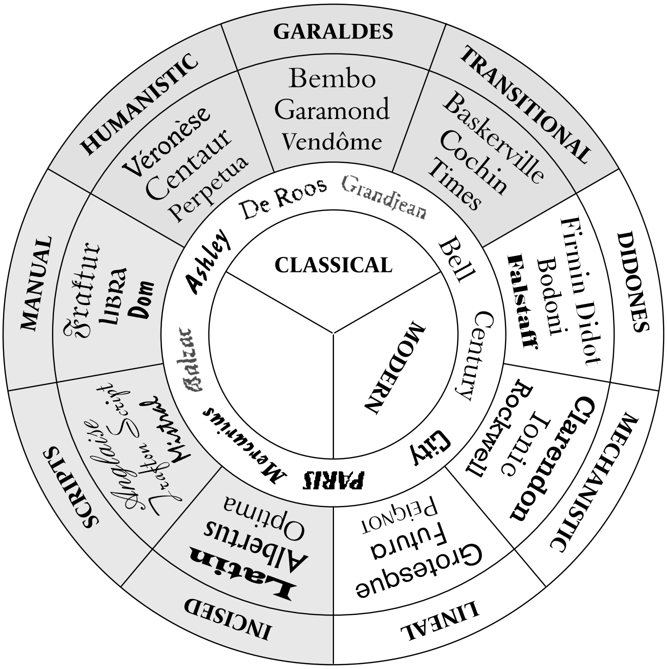


Fig. 3.3 The Vox classification of typefaces, released in 1963 (figure taken from [24, p. 409])

weil es anders n i c h t m ö g l i c h ist Worte auszuzeichnen”$_{\mathrm{DE}}$ (one letterspaces whenever there is no other way of emphasizing) [64, p. 223].
- Baseline/$^{\text{superscript}}$/$_{\text{subscript}}$. used mainly in mathematics. Superscripts should *not* to be used in English ordinal numbers (§9.6). The situation is different for ordinal numbers and abbreviations in French [36, pp. 7–11] where superscript $^{\text{e}}$ (or $^{\text{er}}$ for the masculine first and $^{\text{re}}$ for the feminine first) is used for ordinals: “la 1$^{\text{re}}$ année du XIX$^{\text{e}}$ siècle”$_{\mathrm{FR}}$, and some abbreviations are used (such as “n$^{\text{o}}$”$_{\mathrm{FR}}$ or “n$^{\text{os}}$”$_{\mathrm{FR}}$ for “numéro”$_{\mathrm{FR}}$, “M$^{\text{me}}$”$_{\mathrm{FR}}$ for “Madame”$_{\mathrm{FR}}$, etc.).
- Font size variation is sometimes used to visually mimic prosody:
  Quoth the raven nevermore...
- Typeface choice. This is one of the main connotative variables. In Figure 3.3, the reader can see the 1963 Vox classification of typefaces [24, p. 409]. Each typeface class connotes a historical period, a mood, or an attitude. The Vox classification has been criticized[4] lately because of its blatant Eurocentrism.

A clever choice of typefaces can be seen in the left side of Figure 3.4, where the name of the town Bonn appears thrice in three different typefaces to connote three aspects of the, at that time, German capital: “Politics, parliament, diplomacy” in a bold *Helvetica* typeface, connoting political power by its boldness and simplicity; “music, museums, theaters” in a 19th century *Didot* typeface, connoting culture and

4 https://perma.cc/H259-C77X.

Fig. 3.4 On the left: an advertisement for the city of Bonn [64, p. 317]. On the right: a 1994 Austrian newspaper advertisement: *Go vote!* Others do it too. [52, p. 175]

arts; "Rhine, romanticism, hospitality" in a 19th century broken script, connoting the *Sturm und Drang* movement and romanticism [64, p. 317].

A typeface may also contribute to ideological/political meaning construction. To illustrate this, consider the right side of Figure 3.4. In the upper part, "Go to vote!" in black *Helvetica* exhorts the reader to go and vote, while in the lower part, "Others do it too." written in broken script refers to extreme-right voters[5].

3.2 Graphemics

In phonology, the definition of a phoneme as its elementary unit is straightforward. This is not the case for the definition of the *grapheme*, the elementary unit of *graphemics*, the emic study of the written modality of language.

According to [5], there have been three approaches to graphemics:

1. *phonocentrism*, which is the denial of the importance of studying the written modality of language. This approach is mainly represented by the French linguist Ferdinand de Saussure and the American linguist Leonard Bloomfield, who wrote in 1933:

[5] In this example, broken script connotes nazism. Ironically, the Nazis forbade the use of broken scripts in 1941, calling them "Jewish letters"… [27, p. 138]. After 1941, using broken script typefaces would be an act of resistance that could lead a printer directly to the concentration camp, so one can hardly qualify broken scripts as being nazi. For the opposition between broken scripts and Roman script, see also [64, pp. 216–326].

> writing, of course, is merely a record of speech. […] Writing, like telegraphy or short-hand, is an activity that deals with language, but it is quite different, far less practiced and in-grained, far more superficial in our make-up than speech." [7, pp. 148–149]

To place writing on the same level as telegraphy is an extreme position to which not many people adhere today;

2. *phonographism* (also called *dependence hypothesis*), which states that writing deserves to be studied, but always in connection to speech—see the various arguments in [41, pp. 26–28]; and
3. *autonomism* (also called *autonomy hypothesis*), which states that the written modality of language is of equal importance as the other two modalities (oral and signed) and can be studied independently of them—see the various arguments in [41, pp. 28–31].

The main proponent of autonomism in France, Jacques Anis, defines three kinds of *graphemes*:

1. *alphagrams* are used to build graphemic words. They are obtained by opposition (similarly to phonemes in phonology);
2. *topograms* either regulate first articulation (in this case, they are spaces and punctuation signs) or modify alphagrams (typographic variables such as italic, bold, small caps, etc.);
3. *logograms* are equivalent to sequences of alphagrams ('&', '§', '$', '€', mathematical symbols, abbreviations, sigla, logotypes, etc.).

He also defines two kinds of alphagrams: *nodes*, which are the kernels of graphemic syllables, and *sates* (from the word "satellites"), which constitute the periphery of graphemic syllables. The former corresponds to vowels in phonology and the latter to consonants.

Anis's approach is purely combinatorial, as he presupposes no knowledge of the correspondence of graphemes with phonemes. Opposition is realized through minimal pairs, with the following condition: if the meaning of two mesographetic sequences differing only by one γ-graph is different (no matter how these sequences are rendered phonetically), then these two γ-graphs belong to different graphemes.

One could expect that this approach would provide nodes and sates quite different from their phonological counterparts. Interestingly, this is not the case: Thaine & Penn [56] present a method of calculation of "nodeness" and "sateness" of graphemes in which they *invariably match their phonological counterparts*, namely vowels and consonants. This method, called *spectral decomposition*, has been applied to no less than 26 different writing systems, always with satisfactory results. In Figure 3.5, the reader can see the result of spectral decomposition applied to Old, Middle, and Modern English languages. The vertical axis denotes "nodeness" (for positive values) and "sateness" (for negative values). As can be seen in the figure, consonants and vowels behave as expected, except for <u> and <y>: the former is in the middle of the vowel cluster for Old English, very close to consonants in Middle English, and a bit farther but still the closest to the line separating vowels from consonants for Modern English; as for <y>, it behaves as a vowel in Old English and slowly shifts to the consonantic cluster in Middle and Modern English.

3.2.1 Writing Systems and Scripts

A *writing system* is the structure of the written modality of a given natural language. Graphemes are the elementary units of a writing system. Graphemes are materialized by γ-graphs. We will use the <…> notation for graphemes and the |…| notation for γ-graphs.

In many cases, a given γ-graph may represent the graphemes of more than one language. For that reason, we distinguish the notion of *writing system*, which is language-specific, from the notion of *script*, which is an inventory of γ-graphs with similar properties used by one or more writing systems. Therefore, we distinguish between, e.g., the English writing system and the Roman script. The English writing system uses the Roman script, and so is the German writing system. But the grapheme <ß>, for example, belongs to the German writing system and not the English one. Sometimes γ-graphs of a given script stand for the same grapheme in one writing system and for different graphemes in another writing system—a typical example is the pair of graphs |I| and |İ|. The latter is an esthetic variant of the former in the English writing system, but in the Turkish writing system, <I> and <İ> are different graphemes.

Graphemes of a writing system can be *primary* or *secondary*. The primary ones may occur as stand-alone graphemes. The secondary ones always accompany pri-

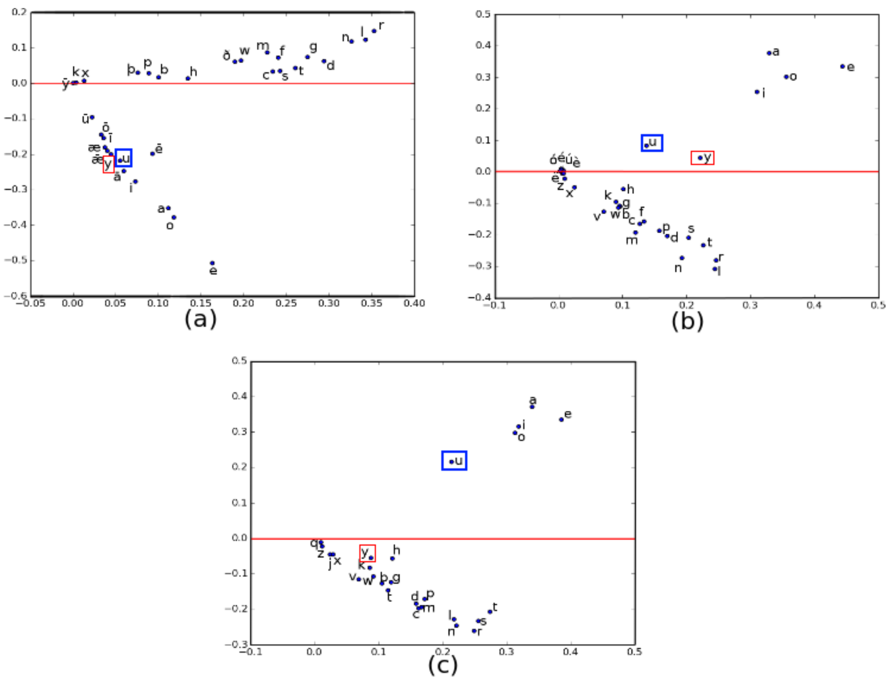


Fig. 3.5 Spectral decomposition of (a) Old, (b) Middle, and (c) Modern English graphemes (figure taken from [56, p. 382])

mary graphemes: for example, in <mañana>, the tilde over the <n> is a secondary grapheme.

To classify writing systems, we will anticipate the next chapter's subject, namely morphology. We can ask: When does meaning emerge in a given writing system? Do individual graphemes carry meaning, or do we need to combine them for meaning to emerge? Typical examples are the grapheme <猫> in the Chinese writing system, which carries the meaning of CAT, and the graphemes <c>, <a>, and <t> in the English writing system, that carry no meaning per se, but where the meaning of CAT emerges when we concatenate them as <cat>. We will call the former a *morphographic* or *pleremic* writing system (from the Greek "πλήρης"$_{\text{EL}}$ ⌊full⌋) and the latter a *phonographic* or *cenemic* writing system (from the Greek "κενός"$_{\text{EL}}$ ⌊empty⌋). These characterizations are statistical: in fact, every writing system contains *morphograms* (graphemes that carry meaning) and *phonograms* (graphemes that do not carry meaning, per se). In the English writing system, <a> can be used as a phonogram, inside the word <cat>, or as a morphogram (when it denotes the definite article). Most writing systems of the world share morphograms such as <&>, <@>, <§>, etc.

The reader will notice that in "phonogram," there is "phono-" meaning "voice," even though we haven't mentioned speech at all in the definitions above. Indeed, our goal was to present morphograms and phonograms in the spirit of Anis's autonomistic approach.

Setting this approach aside, we can achieve a finer classification of phonographic writing systems by considering the correspondence between graphemes and phonemes. There are four main types of phonographic writing systems:

1. *alphabetic* ones, in which primary graphemes represent both consonant and vowel phonemes. For example, in <ὅλος>, the primary grapheme <o> corresponds to a vowel phoneme, the primary graphemes <λ> and <σ> (of which |ς| is an allograph) correspond to consonant phonemes, and the secondary graphemes <῾> and <´> are combined with the primary grapheme <o>;
2. *abjad* ones, in which consonant and long vowel[6] phonemes are represented by primary graphemes. Short vowels (or the absence of vowels) are either not represented or represented by secondary graphemes. For example, in <كاتِب>, primary graphemes <ك>, <ت> and <ب> correspond to consonants, the primary grapheme <ا> corresponds to a long vowel, and secondary grapheme <◌ِ> corresponds to a short vowel. The name "abjad" comes from the four first letters of the historical Arabic collating order. In abjad writing systems (Arabic, Hebrew, Syriac, Thaana), when secondary graphemes are omitted, they have to be deduced by the reader from the context through a complex process involving morphology, syntax, and semantics;
3. *abugida* ones, in which vowels (or their absence) following consonants are represented by secondary graphemes, except for one, the /a/ vowel, which is a "default" vowel, and is implied whenever there is no secondary vowel grapheme.

[6] Following [12, pp. 62–63] one could also consider that abjads are moraic writing systems, where long vowel graphemes correspond to two moras and short vowel graphemes to a single mora.

For example in the Hindi writing system (in the Devanagari script), in the word <लेखक> *writer* we have a primary grapheme <ल> corresponding to a consonant, followed by a secondary grapheme <◌े> corresponding to a vowel other than /a/, followed by a primary grapheme <ख> corresponding to a consonant that carries the implicit vowel /a/ and a final primary grapheme <क>.

4. *syllabaries*, in which every grapheme, called a *syllabogram*, corresponds to a syllable. A typical example is the pair of Japanese syllabaries katakana and hiragana, in each of which we have 45 graphemes representing CV syllables and one grapheme representing a C syllable (namely /n/). For example, <ドクトル> "doctor" in katakana consists of four katakana graphemes <ド>, <ク>, <ト> and <ル> corresponding to four CV syllables. The same word in the rōmaji writing system, the official adaptation of the Roman script to the Japanese language, becomes <dokutoru>.

We cannot stress enough that in every writing system, we can find traces of virtually all types of graphemes. For example, the cassette tape was frequently abbreviated as <K7> in France for readers old enough to have witnessed it. In <K7>, the first grapheme is pronounced /ka/, the name of the letter <K> in the French writing system, and is a syllabogram. In contrast, the second grapheme <7> is pronounced /sɛt/, the French word for "seven." Both graphemes are used as *autonymic rebus* (use of a letter's or symbol's name)[7].

Also, it should be noted that not all languages using a given script use it similarly. The best example is *Uyghur*: while all other Arabic-script-based writing systems (such as Arabic, Persian, Urdu, Sindhi, etc.) are abjad, the Uyghur writing system became alphabetic, after several orthographic reforms [22].

The basic units of the Chinese script are called *sinograms*. A given sinogram carries both meaning and phonemic information, which we call *semanticity* and *phoneticity*. See § 3.2.6 for more information. ☞ 55

3.2.2 Pictography, Emoji

According to Wikipedia, a *pictogram* is a graphic symbol that conveys meaning through its pictorial resemblance to a physical object. There is an ISO Standard [32] that contains some 163 pictograms that we can encounter worldwide (airplanes, banknotes, babies with diapers, and the like). But are pictograms language?

As [11, pp. 196–201] notices, pictograms constitute a *pictorial sign system*. Some of them may obtain linguistic interpretations and end up being morphograms in a given writing system (this is the case of the Chinese morphogram "川"$_{\text{ZH}}$ ⌊river⌋ that still today evokes a river, or "山"$_{\text{ZH}}$ ⌊mountain⌋, or "人"$_{\text{ZH}}$ ⌊person⌋). Using pictograms inside text is a creative writing process, dear to modernist authors such as Ezra Pound or Peter Handke (Nobel Prize in Literature 2019). Figure 3.6 displays an excerpt

[7] The same principle has been applied to the name of the Doctor's faithful companion K9, a pun on the pronunciation of "canine." This electronic dog has served the Doctor from 1977 to 2010. NASA is preparing an autonomous K9 robot for a future Mars mission https://perma.cc/SJ3D-XDZ3.

from an English translation of Peter Handke's *The Goalie's Anxiety at the Penalty Kick*, published in 1970. The example is interesting because it illustrates a three-step process: first, words are "alienated" by double-quoting; then, they are substituted by pictograms, the meaning of which can be deduced from the verbal context; and finally, they are replaced by sequences of pictograms occupying the whole line. The paragraph can be considered a multimodal representation of the goalie's vision—once the vision is over, the protagonist closes the curtains and leaves the room, a perfectly theatrical ending.

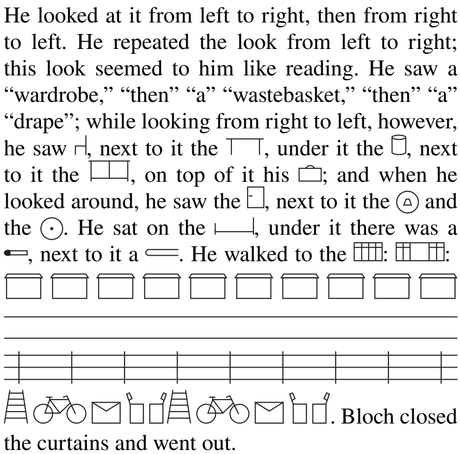


Fig. 3.6 Excerpt from Peter Handke's *The Goalie's Anxiety at the Penalty Kick*

The use of pictography in the text would be pretty anecdotic in the absence of a new trend in digital communication, called *emoji*. The Oxford Dictionary has attributed to the emoji 😂 the status of "word" by calling it "Word of the Year 2015," Nevertheless, in linguistics, emoji are not considered as being part of language—after all, they have no first and second articulation, even though some combinatorics exists. They are mostly considered graphical signs interacting with language [37]. Indeed, they can substitute parts of a sentence or be apposed at the end of a sentence. In the latter case, they are not part of the syntax tree of the sentence [10].
MIT Press has published a book by the Chinese artist Xu Bing [65, 8], entirely written in emoji and Roman-script punctuation. The book's title is also in emoji; "• → 🚶 → •", the verbal representation of this title is *From Point to Point. Book from the Ground*, and this is how the book appears in bibliographical databases. In Figure 3.7, we display a short paragraph from this book. In it, the main character 🚶 wishes to move from place W to place E; 🚶 wonders whether 🚶 should take the bus or the subway; 🚶 consults an app, which informs 🚶 that there is a traffic jam, and that 🚶 should not take the bus but rather the subway; so 🚶 takes the subway and sees (the action of seeing being represented by the pictogram of an eye followed by a dotted arrow) a crowd of people… Turning this sequence of pictograms into a narrative thread would not be possible without the use of punctuation, including special paren-

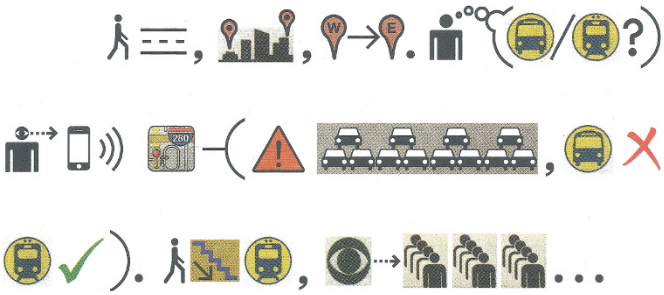


Fig. 3.7 An excerpt (page 7) from Xu Bing's •→†→• (*Book from the Ground*)

theses for thoughts (using bubbles as in comic strips) and for indirect speech (the message provided by the app). Also, merging identical emoji is a graphical method used to represent plurality. This attempt to write a book entirely in emoji is still an isolated attempt, and no study has yet evaluated its understandability, reading speed, universality, etc. But it is an interesting step in this new direction.

3.2.3 Orthography

Orthography, in which "ὀρθός"$_{\text{EL}}$ stands for "straight, correct," is a prescriptive activity where competent authorities request writers to use specific word spellings. Tools like *spellcheckers* suggest corrections to text based on orthographical rules. The identity of authoritative sources is a political issue: in Germany, it is the "Rat für deutsche Rechtschreibung"$_{\text{DE}}$ ⌊Council for German Orthography⌋, an international body financed by seven German-speaking countries; in France, it is the "Académie française"$_{\text{FR}}$ ⌊French Academy⌋, founded by royal decree in the 17th century. Both institutions monitor the evolution of written language and emit recommendations—if not followed, these recommendations are withdrawn. In the English-spoken area, there is no official authority. Orthography is regulated by dictionary publishers such as Merriam-Webster [42], named after Noah Webster, whose first dictionary (1806) [63] is at the origin of most US-specific English spellings, such as "color," or "center."

Among the tools for political power at the disposal of regimes are *orthographic reforms*. In the twentieth century, there has been a multitude of such reforms, such as the 1928 Kemal Atatürk reform of the Turkish writing system (moving from the Arabic to the Roman script) [19], the 1930 and 1941 Soviet reforms of the Mongolian writing system (first moving from Mongol script to Roman script, and then from Roman to Cyrillic), the disastrous 1982 Greek monotonic reform [26] that stripped most secondary graphemes from the Greek writing system and many more. Although this will probably not happen for Greek, some reforms do happen to be revocated years later when the political situation allows it. This is the case of Malayalam, where a governmental attempt to "simplify" the script is progressively abandoned. Thanks to modern font technologies, the writing system returns to its previous, traditional state [38].

In the remainder of this section, we will first examine the graphemic phenomena of hyphenation/non-breakability and (graphemic) gender-neutral writing, and we will then conclude with a closer study of the Chinese writing system.

3.2.4 Hyphenation and Non-breakability

According to legend, an RCA Marketing Manager
received once a nightly phone call from a
disturbed customer.
His 301 had just hyphenated “God.”

Paul E. Justus, *There's More to Typesetting Than Setting Type*

Hyphenation breaks a graphemic word into two parts, adds a hyphen to the first part, which typically ends a line, and places the second part on the next line. The necessity of breaking a word is of a graphetic nature: it depends on the room left at the end of a line and the flexibility of interword spaces of that line. But the *way* a word is broken is graphemic and involves several, if not all, levels of linguistics.

Every writing system has its own hyphenation rules, and interestingly, the term used for this operation in each reveals their linguistic nature. In Italian, it is called “sillabazione”$_{\mathrm{IT}}$ ⌊syllabification⌋, as Italian hyphenation is purely phonetic, so that words can be broken only at syllable boundaries. In French, it is called “césure”$_{\mathrm{FR}}$ ⌊cut-off⌋, a rather neutral term. Indeed, French hyphenation is mainly syllabic but can also be morphological, as in “inter-action”$_{\mathrm{FR}}$. In German, hyphenation is called “Worttrennung”$_{\mathrm{DE}}$ ⌊word separation⌋ and respects the component structure: at first, components are separated; if this is not sufficient, then a syllabic hyphenation can be attempted in the last component, and finally, as a last resort, other components are allowed to be hyphenated. This results in a stratification of potential hyphenation points in three levels, as illustrated in the following example:

(3) *Wahr-*$_1$ *schein-*$_3$ *lich-*$_3$ *keits-*$_1$ *rech-*$_2$ *nung*

the German word for “probability.” In the above, the 1s are the recommended breakpoints, the 2 is a second-level breakpoint, and the 3s are to be used only as a last resort. German hyphenation is special also because, at least before the 1996 orthographical reform, graphemes were substituted during hyphenation: a <ck> digraph was hyphenated as <k-k> (the word “backen”$_{\mathrm{DE}}$ would be broken as “bak-ken”$_{\mathrm{DE}}$) and reduced triple consonants were expanded (the word “Schiffahrt”$_{\mathrm{DE}}$, without <ff> ligature, is built out of the components “Schiff”$_{\mathrm{DE}}$ and “Fahrt”$_{\mathrm{DE}}$, but the three <f>s are reduced into two—when broken, the double <f> of “Schiff”$_{\mathrm{DE}}$ is re-established: “Schiff-fahrt”$_{\mathrm{DE}}$).

English goes one step further as its hyphenation is morphological, including for grammatical morphemes: “amazing” will be hyphenated as “amaz-ing,” contradicting phonological syllable boundaries. For some words, hyphenation depends on the part of speech, e.g., “record” will be hyphenated “rec-ord” when a noun and “re-cord” when a verb.

Finally, in all languages featuring hyphenation, some word breaks have to be avoided because the first part of the word can attract the reader’s attention and disrupt the reading process. For example, in French, breaking the prefix “con-”$_{\text{FR}}$ is forbidden as it is an offending word. A break such as “the-rapist” in English may be confusing [34, p. 449]. In German, “Spargel-der”$_{\text{DE}}$ (instead of the correct “Spar-gelder”$_{\text{DE}}$) mislead the reader into considering “Spargel”$_{\text{DE}}$ ⌊asparagues⌋ as being the first morpheme, instead of “Spar”$_{\text{DE}}$ ⌊savings⌋.[8]

Non-breakability is avoiding breaking between specific graphemic words. It serves the same goal: to maintain a continuously flowing reading process. For example, one will not break between an abbreviated word and a number, as in “p. 127” (“page 127”), or between a proper name’s initial and the family name, as in “T. Waits.”

3.2.5 Graphemic Gender-neutral Methods

There has been a steadily increasing effort to make language gender-neutral in the last decades. Among the various methods to achieve this goal (paraphrases, gender-neutral pronouns, etc.), some are graphemic methods in the sense that they are based on graphemic substitution or suffixes.

German [28, pp. 58–61] has been pioneering in the field by introducing, already in 1981, a capital <I> in the feminine suffix of bi-gender nouns, such as “StudentInnen”$_{\text{DE}}$ ⌊students$_{♂\text{ or }♀}$⌋ or “KollegInnen”$_{\text{DE}}$ ⌊colleagues$_{♂\text{ or }♀}$⌋. When this happens, the surrounding adjectives take a feminine form, as in “jede neue FreundIn”$_{\text{DE}}$ ⌊every$_{♀}$ new$_{♀}$ friend$_{♀}$⌋. This method is called “binnen-I”$_{\text{DE}}$. It has been criticized for its preference for feminine forms [35, p. 217] and for the difficulty of perceiving the camerality (i.e., the case, as in upper and lower case) of the <I> grapheme [45].

French [28, pp. 61–69] uses a different method called “écriture inclusive”$_{\text{FR}}$ ⌊inclusive writing⌋. It consists of adding a centered dot <·> after the masculine form, followed by the suffix of the feminine form, e.g., “étudiant·e”$_{\text{FR}}$ ⌊student$_{♂\text{ or }♀}$⌋. In the case of a plural form, an additional centered dot is added, followed by the plural suffix, e.g., “ami·e·s”$_{\text{FR}}$ ⌊friends$_{♂\text{ or }♀}$⌋. One of écriture inclusive’s rules is alphabetic collation order for choosing the suffix that stays with the stem. In most cases, this gives the expected result (“voisin·e”$_{\text{FR}}$ since “voisine”$_{\text{FR}}$ comes lexicographically after “voisin”$_{\text{FR}}$). Still, it produces hilarious results in some cases, such as “jalouse·loux”$_{\text{FR}}$… See Exercise 8-4, on p. 271 for an implementation of écriture inclusive.

As for Spanish and Italian, their morphologies [28, pp. 50–58] are optimally suited for graphemic gender-neutral writing since the difference between mascu-

[8] Duden online, https://perma.cc/6ZSW-2Z3Z.

Fig. 3.8 On the top: France, 2018, “All$_{♂\text{ or }♀}$ against the law”, the word “TOU·T·E·S”$_{\text{FR}}$ carrying *three* centered dots instead of two, as a result of its author’s uncertainty on the proper gender-neutral version of “TOUS”$_{\text{FR}}$ ⌊(⌋which is “TOUT·E·S”$_{\text{FR}}$). Lower left part: Argentina, 2022, “All$_{♂\text{ or }♀}$ against patriarchy” (Clarín, October 7th, 2022). Lower right part: Austria, 2015, “Dear dog owners$_{♂\text{ or }♀}$, please be considerate of the need for peace of the neighbors$_{♂\text{ or }♀}$...” (Wikimedia).

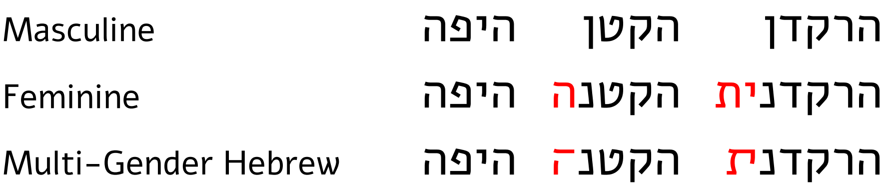


Fig. 3.9 The sentence “The beautiful little dancer” in masculine, feminine, and gender-neutral form Hebrew, as designed by Michal Shomer. Gender-specific suffixes are colored in red.

line and feminine forms often consists of just one grapheme. In Italian, for example, we have “ragazzo”$_{\text{IT}}$/“ragazza”$_{\text{IT}}$ (boy/girl) and “ragazzi”$_{\text{IT}}$/“ragazze”$_{\text{IT}}$ (boys/girls). A popular graphemic gender-neutral method consists of replacing the graphemes that mark gender with a non-alphabetic grapheme such as <@> or <*> (using <@> also has a symbolic side since it looks like an <a> enveloped in an <o>, and <a>, <o> are precisely the graphemes marking feminine and masculine gender in the singular). In Spanish, where the gender markers are only <a> and <o>, the letter <e> has been used as a gender-neutral marker, the flag word being “todes”$_{\text{ES}}$ ⌊everyone$_{♂\text{ or }♀}$⌋, instead of “todas”$_{\text{ES}}$ ⌊all$_{♀}$⌋ and “todos”$_{\text{ES}}$ ⌊all$_{♂}$⌋.

The reader can see examples of the French, Spanish, and German methods in Figure 3.8.

In Israel, a young font designer, Michal Shomer, has started a beautiful project called *Multi Gender Hebrew* on gender-neutralizing Hebrew script by creating new graphemes [53]. The new graphemes replace those used in gender-specific affixes. An example of her work can be seen in Figure 3.9.

1		*héng*	三	14		*héng-zhé*	口	27		*héng-zhé-tí*	鳩
2		*tí*	虫	15		*héng-piě*	又	28		*héng-zhé-gōu*	丹
3		*shù*	中	16		*héng-gōu*	写	29		*héng-xié-gōu*	凬
4		*shù-gōu*	小	17		*shù-zhé*	山	30		*shù-zhé-zhé*	亞
5		*piě*	八	18		*shù-wān*	四	31		*shù-zhé-piě*	专
6		*wān-piě*	大	19		*shù-tí*	民	32		*shù-ān-gōu*	儿
7		*shù-piě*	厂	20		*piě-zhé*	公	33		*héng-zhé-zhé-zhé*	凸
8		*diǎn*	主	21		*piě-diǎn*	巛	34		*héng-zhé-zhé-piě*	及
9		*nà*	人	22		*piě-gōu*	乂	35		*héng-zhé-wān-gōu*	亄
10		*diǎn-nà*	仚	23		*wān-gōu*	豖	36		*héng-piě-wān-gōu*	阝
11		*píng-nà*	走	24		*xié-gōu*	弋	37		*shù-zhé-zhé-gōu*	丂
12		*tí-nà*	夊	25		*héng-zhé-zhé*	凹	38		*héng-zhé-zhé-zhé-gōu*	乃
13		*tí-píng-nà*	辶	26		*héng-zhé-wān*	朵	39		*quān* o	[illegible]

Fig. 3.10 The 39 fundamental strokes of sinograms (number, shape, name, example of use).

3.2.6 Sinographemics

> 鬱 is a Chinese character meaning “depression” which, depressingly, has 29 strokes.
>
> Masato Hagiwara, *Real-world NLP*

Sinographemics is the study of the Chinese writing system and, more generally, of the Chinese-origin characters used in Japanese (kanji), in Korean (hancha), and historically in Vietnamese. To avoid misleading terms such as “ideograms” and “logograms,” we call these graphemes *sinograms*.

Between 1952 and 1964, there was a simplification of 2,236 sinograms in mainland China. Today, we have two variants of the Chinese writing system: *simplified* Chinese, used in mainland China, and *traditional* Chinese, used in Taiwan, Hong Kong, and Macau. The difference is sometimes dramatic, as in the case of the words “leaf”: “叶”$_{ZH}$ (simplified) vs. “葉”$_{ZH}$ (traditional).

The structure of sinograms is remarkable: to draw the more than 100,000 existing sinograms, one uses an inventory of only 39 *calligraphic strokes*, displayed in Figure 3.10. Frequent groups of neighboring strokes are called *constituents* [43, p. 11]. Chinese scholars have selected some of these constituents to sort sinograms in dictionaries. They are called *radicals*. The best-known example is the dictionary *Shuowen*, compiled by the scholar Xu Shen in the 2nd c. CE.

There are several ways of combining constituents: horizontally, vertically, surrounding, etc., in a recursive way. By repeatedly combining constituents, one can end up with quite complex sinograms—a well-known example is “𰻝𰻝”$_{ZH}$ ⌊verbose⌋, which combines four “龍”$_{ZH}$ ⌊dragon⌋ sinograms, and therefore contains 64 strokes [14, p. 75].

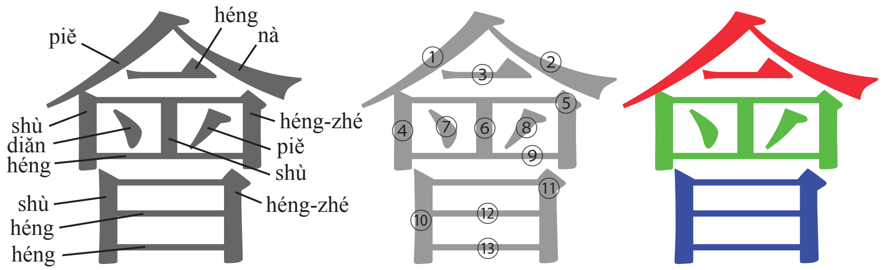


Fig. 3.11 Sinogram 會, the ancient spelling of the name of the town Aizu-Wakamatsu in Japan. On the left are the names of the thirteen strokes. In the center: stroke drawing order. On the right: the three constituents of 會

In Figure 3.11, the reader can see the ancient spelling of Aizu-Wakamatsu in Japan. As seen on the right, the sinogram has three constituents, the bottom one being the Shuowen radical 曰 TO SAY. According to [33], the sinogram “會”$_{JA}$ can be explained as “To 亼 (meet) around the 罒 (fire) and to 曰 (chat)”.

Constituents contribute in different ways to the information stored in the sinogram. In a case such as “休”$_{ZH}$ ⌊rest⌋, both constituents contribute to the semantics of the sinogram since the left one stands for a man “人”$_{ZH}$, and the right one stands for a tree “木”$_{ZH}$, and standing close to a tree is indeed a way of resting. The case of “蟬”$_{ZH}$, cicada, is different: the left constituent “虫”$_{ZH}$ has the meaning of worm. Still, the right constituent “單”$_{ZH}$ is purely phonetic (“chán”) and determines the phonetic representation of “蟬”$_{ZH}$. In other words, “蟬”$_{ZH}$ is a “worm” pronounced “chán” like “單”$_{ZH}$, and therefore a cicada. Taken the other way around, among all sinograms pronounced “chán,” this is the one that belongs to the category of worms and represents, therefore, a cicada.

More generally, constituents of sinograms have varying amounts of *semanticity* (i.e., the amount of contribution of their meaning to the meaning of the global sinogram) and *phoneticity* (i.e., the amount of contribution of their phonetic realization to the phonetic realization of the global sinogram) [21, 51, 25].

Let us now give an example of how Chinese semanticity and phoneticity have been inherited in the Japanese writing system. The sinogram “任”$_{ZH}$ means “responsibility” both in Japanese and Chinese. Its right constituent “壬”$_{ZH}$ originally meant “burden” and has high semanticity as a constituent of “任”$_{ZH}$. The left constituent, “人”$_{ZH}$ has high phoneticity. Indeed, in Mandarin Chinese, “人”$_{ZH}$ is “ren^2” and “任”$_{ZH}$ is “ren^4,” so that both are realized by the same syllable with only a change of tone. The Japanese language has adopted the Chinese readings of sinograms as they were imported in Japan in the 1st century CE. These Chinese-originating readings are called *on* in Japanese. Original Japanese words were also attached to the imported Chinese sinograms and are now readings of the same sinograms. They are called *kun* in Japanese. In the case of “任”$_{ZH}$, because of the differences between Chinese and Japanese phonology and the fact that Japanese is not a tonal language, both “rén” and “rèn” became “nin” in Japanese, so that in *on* the sinograms “人”$_{JA}$ and “任”$_{JA}$ are both read “nin”. Their *kun* readings, namely “hito” and “makaseru,” are completely unrelated to each other.

We will conclude this section with an example of the subtle interaction between phonology and graphemics in translating foreign brand names. Let us take the example of *Coca-Cola*, which has been translated “可口可樂”$_{ZH}$ in traditional Chinese.[9] This sinographic sequence is read “ke^{3}kou^3 ke^{3}le^4,” which is quite close to the American /ˈkoʊkə ˈkoʊlə/. But the reason for the choice of the specific sinograms among all homophones is that, taken individually, the four sinograms have the meanings “can,” “mouth,” “can,” and “music,” and taken in groups of two they mean “delicious” and “cola pastry.” As a result, the logo acts both on the phonemic (ke^{3}kou^3 ke^{3}le^4) and on the graphemic level (“delicious cola pastry”). And due to the inheritance of Japanese sinogram semantics from Chinese, a speaker of Japanese will interpret “可口可樂”$_{JA}$ as “singing pleasure” since the first group of two sinograms means “mouth” and the second “pleasure” in the Japanese writing system.

3.3 Psycholinguistic Aspects of Reading

In this section (heavily inspired by [61]), we will deal with the psychological aspects of reading. Visual analysis is a process that can be subdivided as follows:

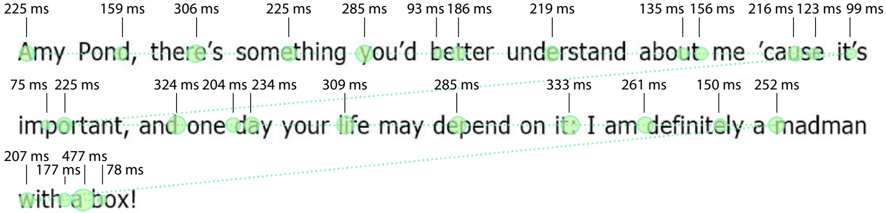
- first we transfer the visual input into some sort of buffer,
- then information is transferred to working memory,
- and finally, the analyzed input is integrated with the linguistic and cognitive interpretation of the text.

We need only 50 ms to recognize a word, but when we read, we need as much as 250 ms per word because transferring information from the visual buffer to working memory takes time. Eye movements are of two sorts: *fixations* and *saccades* (jumps).

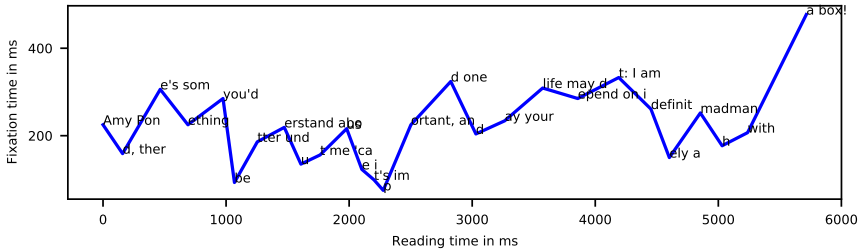
In Fig. 3.12(a), we see a sentence on which we have added fixation points as monitored by a very advanced device (an *EyeBrain T2** oculometer [47]). The eye’s peripheral vision prepares for the next saccade at every fixation. The eye will identify the beginning graphemes of the following words, recognize graphetic features of additional graphemes, and finally, estimate the length of subsequent words to plan for the next fixation point. The reader may notice that when saccading from the first to the second line, the eye is fixating on the letter “o” in the middle of the word “important,” and then goes even a bit farther to letter “p,” before moving right—the same happens between the second and the third line: first the “a” is read, and then the “w” of “with,” and only after that it moves to the interval between “a” and “box”. The exact movement of the eye is shown in part (c) of the figure (the horizontal axis is time, and the vertical axis corresponds to the horizontal dimension of the text). Finally, in part (b) of the figure, we have displayed the fixation time vs. the reading time—one observes that the part “better understand about me ’cause it’s” has significantly shorter fixations than the part “one day your life may depend on

[9] https://perma.cc/LM2G-ZJYG

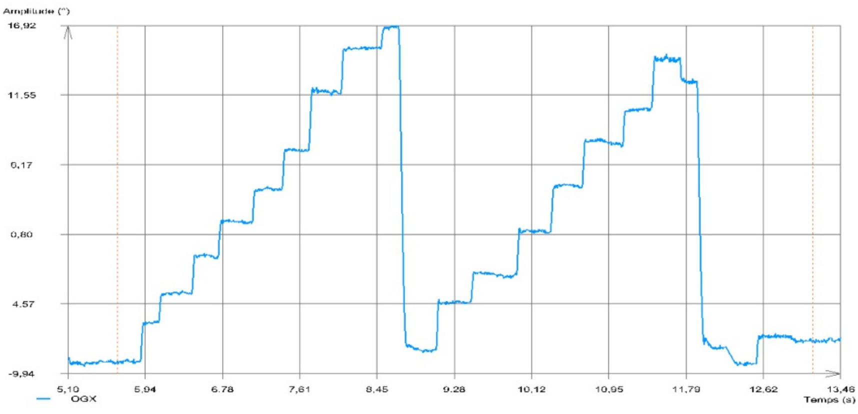
it." This can be because it is the second line or because words are shorter. It can also be related to the semantics of the text. It would be interesting to correlate it with the speed of the spoken utterance.



(a)



(b)



(c)

Fig. 3.12 (a) A quotation of the 11th Doctor (*The Eleventh Hour*, S31:1) read by Nassim (35 years old, not English-native speaker) whose left eye was monitored by an *EyeBrain T2** oculometer device [47] with a data measurement frequency of 300 Hz. The fixation locations and times have been added to the original text.
(b) Fixation times vs. global reading time. One can notice that the two shortest fixation points are at word begins with a second fixation in the same word: "*be*tter" and "im*p*ortant." In the case of "'cause" we have two relatively long fixations in the same word ('ca*us*e) probably because the shortened form of the word is not familiar to the reader.
(c) Movement of Nassim's left eye vs. time: we see that when reading the third line he first targets the blank space between "with" and "a" and only after 177 ms shifts to the left margin.

3.4 Further Reading

3.4.1 Literature

The literature is quite limited as mainstream linguists excluded graphemics for most of the twentieth century. The first book that dared stand up to the Saussurian and Bloomfeldian linguists was Gelb's *A Study of Writing* [17], published in 1952. It is still a valuable read, despite some expressions that may be out of place today and a tendency to consider alphabetic writing as the *nec plus ultra* of writing. In the chapter "Future of Writing" he states: "What we should look for is a system of writing combining the exactness of the IPA alphabet with the formal simplicity of a shorthand system." IPA is phonetic and not phonological, which means that the writing system he proposes would faithfully reflect the regional accent of each speaker… A nightmare!

The impressive volume [13] synthesis of the writing systems of the world written by different authors for each system. A much shorter overview of the world's writing systems is [44], which has the advantage of showing real newspapers in the corresponding writing systems.

The best modern introductions to grapholinguistics are Coulmas's [11], Meletis & Dürscheid's [41], and for readers of German, Dürscheid's [15], already at its 5th edition. Those interested in the linguistic aspects of typography should read [64] (again in German). The publisher *Fluxus Editions*[10] is specialized in grapholinguistics and published, among other books, the proceedings of the biennial conference *Grapholinguistics in the 21st Century*. Those interested in typography and design should consult the site of ATypI (Association Typographique Internationale). There is also conference ICTVC[11] and journals like *Visible Language*[12]. On the practical side, an extensive description of tools for drawing and using typefaces and a short history of typography can be found in [24].

3.4.2 LaTeX

Two extensions of the TeX engine are capable of efficiently typesetting in any of the world's languages: XeTeX and LuaTeX. This book has been typeset with the former and the packages *polyglossia*[13] for managing the specificities of each language and *fontspec*[14] for using various types of fonts. For the proper use of sinographic languages, the package *xeCJK*[15] should be used.

[10] http://www.fluxus-editions.fr
[11] https://ictvc.org/2022/en/
[12] https://journals.uc.edu/index.php/vl
[13] https://ctan.org/pkg/polyglossia
[14] https://ctan.org/pkg/fontspec
[15] https://ctan.org/pkg/xecjk

3.4.3 Science Fiction

Science Fiction and Fantasy often use exotic writing systems, even though these are mostly imitating Western alphabets. There is a nice blog[16] on fictional alphabets (including a section for Whovians) by the CREWS project at Cambridge. For graphemic methods used in Science Fiction, and more generally in speculative fiction, see [29]. For a beautiful account of typography in Science Fiction movies, see [1].

3.5 Exercises

Exercise 2-1: Evaluating ALA-LC Transcriptions of Arabic and Greek

Download[17], read carefully and evaluate the American Library Association / Library of Congress (ALA-LC) "romanizations" of Arabic and Greek.

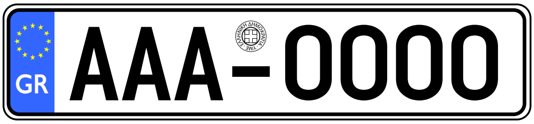
Exercise 2-2: Graphotactics of English

Explore the graphotactics of English. Rank the following Whovian named entities in decreasing order of conformance to English graphotactics:

Lethbridge, Zygon, Dalek, Angstrom, Graham, Yrcanos, Yaz, Pting, Zaakros, Susan.

Exercise 2-3: Greek Car License Plate and Signs

A modern Greek car license plate looks like this:



where "A" can be any letter among the following: "A," "B," "E," "H," "I," "K," "M," "N," "O," "P," "T," "X," "Y," "Z." What is the particularity of this set of letters? Are the letters on the license plate graphemes of the Greek writing system, graphemes of the English writing system, or something else?

What about the following images:

[16] https://crewsproject.wordpress.com/category/writing-systems/fictional-alphabets/

[17] https://www.loc.gov/catdir/cpso/romanization/arabic.pdf and https://www.loc.gov/catdir/cpso/romanization/greek.pdf

1. 

2. 

3. 

4. 

Exercise 2-4: Predictability of New Sinograms

How predictable are sinograms added to the Unicode standard? Take sinograms of the Basic Multilingual Plane (BMP), analyze their structure, and find the probabilities of component occurrence for each position in each structural pattern. Then, find the most probable sinograms according to these probabilities and check whether they have indeed been included in the Supplementary Ideographic Plane (SIP). ☞ 345

This decomposition shall be purely combinatorial. No information on the semantics of the components will be used.

Predict the 1,000 most probable sinograms out of the information contained in the BMP and count those contained in the SIP.

Exercise 2-5: Exotype Classification

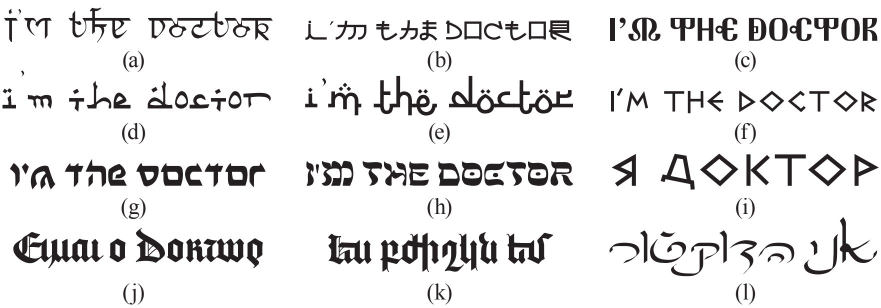


Fig. 3.13 Examples of exotypes

The term *exotype* has been coined by the French designer Jean Alessandrini in 1979 [3, p. 44], as part of his typeface classification scheme *Codex 80*. An exotype is a typeface for script S1 that *simulates* script S2. The reader can see some examples on Fig. 3.13: (a) S1 = Latin & S2 = Devanagari, (b) S1 = Latin & S2 = Japanese, (c) S1 = Latin & S2 = Glagolitic, (d) and (e) S1 = Latin & S2 = Arabic, (f) S1 = Latin & S2 = Epigraphical Greek, (g) and (h) S1 = Latin & S2 = Hebrew, (i) S1 = Cyrillic & S2 = Epigraphical Greek, (j) S1 = Greek & S2 = Gothic, (k) S1 = Armenian & S2 = Gothic, (l) S1 = Hebrew & S2 Arabic.

How would you classify exotypes?

References

[1] Dave Addey. *Typeset in the Future: Typography and Design in Science Fiction Movies*. Abrams, 2018.

[2] Michael H. Adler. *The Writing Machine. A History of the Typewriter*. London: George Allen & Unwin, 1973.

[3] Jean Alessandrini. "Une nouvelle classification typographique: le Codex 1980". In: *Communications & Langages* 43 (1979), pp. 35–56.

[4] Hans Peter Althaus. "Graphetik". In: *Lexikon der germanistischen Linguistik*. Ed. by Hans P. Althaus, Helmut Henne, and Herbert E. Wiegand. Tübingen: Niemeyer, 1980, pp. 105–118.

[5] Jacques Anis. *L'écriture, théorie et descriptions*. Bruxelles: De Boeck, 1988.

[6] Robert Basil. "Graphology and Personality: 'Let the Buyer Beware'". In: *Skeptical Inquirer* 13 (1989), pp. 241–243.

[7] Leonard Bloomfield. *Language*. London: Allen and Unwin, 1933.

[8] Mathieu Borysevicz, ed. *The Book about Xu Bing's Book from the Ground*. The MIT Press, 2014.

[9] Richard G. Casey and Éric Lecolinet. "A survey of methods and strategies in character segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.7 (1996), pp. 690–706.

[10] Neil Cohn, Jan Engelen, and Joost Schilperoord. "The grammar of emoji? Constraints on communicative pictorial sequencing". In: *Cogn. Research* 4 (2019).

[11] Florian Coulmas. *Writing Systems: An Introduction to Their Linguistic Analysis*. Cambridge Textbooks in Linguistics. Cambridge: Cambridge University Press, 2002.

[12] Peter T. Daniels. *An exploration of writing*. Bristol: Equinox, 2018.

[13] Peter T. Daniels and William Bright, eds. *The World's Writing Systems*. Oxford University Press, 1996.

[14] John DeFrancis. *The Chinese Language, Facts and Fantasy*. Hawaii: University of Hawaii Press, 1984.

[15] Christa Dürscheid. *Einführung in die Schriftlinguistik*. 5th ed. Göttingen, Bristol: Vandenhoeck & Ruprecht, 2016.

[16] Adrian Furnham and Barrie Gunter. "Graphology and personality: Another failure to validate graphological analysis". In: *Personality and Individual Differences* 8.3 (1987), pp. 433–435.

[17] Ignace J. Gelb. *A Study of Writing*. The University of Chicago Press, 1952.

[18] Gérard Genette. *Paratexts*. Cambridge: Cambridge University Press, 1997.

[19] Lewis Geoffrey. *The Turkish Language Reform: A Catastrophic Success*. Oxford and New York: Oxford University Press, 1999.

[20] Audrey Giles. "Forensic Examination of Documents". In: *Crime Scene to Court: The Essentials of Forensic Science*. Ed. by Peter C. White. Royal Society of Chemistry, 2016, pp. 229–259.

[21] Andreas Guder-Manitius. *Sinographemdidaktik. Aspekte einer systematischen Vermittlung der chinesischen Schrift im Unterricht Chinesisch als Fremdsprache*. Vol. 7. Sino-Linguistica. Tübingen: Julius Groos Verlag, 1999.

[22] Yannis Haralambous. “Breaking Arabic: the creative inventiveness of Uyghur script reforms”. In: *Design Regression* (2021). https://designregression.com/article/breaking-arabic.

[23] Yannis Haralambous. “Drucksatz in gebrochenen Schriften”. TEX-Tagung DANTE 2000, Technische Universität Clausthal, Mar 2000, Clausthal, Germany https://hal.archives-ouvertes.fr/hal-02111827/document. 2000.

[24] Yannis Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O’Reilly, 2007.

[25] Yannis Haralambous. “New Perspectives in Sinographic Language Processing Through the Use of Character Structure”. In: *CICLing 2013: 14th International Conference on Intelligent Text Processing and Computational Linguistics, Samos*. Vol. 7816. Springer LNCS. 2013, pp. 201–217.

[26] Yannis Haralambous. “Phonocentrism in Greece: Side Effects of Two Centuries of Diglossia”. poster presented at *AWLL12*, Cambridge, UK, https://hal.archives-ouvertes.fr/hal-02480230. 2019.

[27] Yannis Haralambous. “Typesetting old german: Fraktur, Schwabacher, Gotisch and initials ”. In: *TUGboat* 12.1 (1991), pp. 129–138.

[28] Yannis Haralambous and Joseph Dichy. “Graphemic Methods for Gender-Neutral Writing”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 41–89.

[29] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. “Graphemic and Graphetic Methods in Speculative Fiction”. In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[30] Richard L. Hopkins. *Tolbert Lanston and the Monotype: The Origin of Digital Typesetting*. Tampa, FL: University of Tampa Press, 2012.

[31] *International Standard Iso 107311–1976. Alphanumeric character sets for optical recognition. Part I: Character set OCR-A. Shapes and dimensions of the printed image*. ISO, 1973.

[32] *International Standard Iso 7001:2023. Graphical symbols — Registered public information symbols*. ISO, 2023.

[33] Bernhard Karlgren. *Grammata Serica Recensa*. Stockholm: Museum of Far Eastern Antiquities, 1957.

[34] Donald E. Knuth. *The TEXbook*. Vol. A. Computers & Typesetting. Addison-Wesley, 1986.

[35] Helga Kotthoff and Damaris Nübling. *Genderlinguistik*. Tübingen: Narr Francke Attempto, 2018.

[36] *Lexique des règles en usage à l’Imprimerie nationale*. 3rd ed. Paris: Imprimerie nationale, 2002.

[37] Lorenzo Logi and Michele Zappavigna. “A social semiotic perspective on emoji: How emoji and language interact to make meaning in digital messages”. In: *New Media & Society* (2021). doi:10.1177/14614448211032965.

[38] Kavya Manohar and Santhosh Thottingal. “Malayalam Orthographic Reforms. Impact on Language and Popular Culture”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 329–351.

[39] Alan Marshall. *Du plomb à la lumière : La Lumitype-Photon et la naissance des industries graphiques modernes*. Paris: Maison des Sciences de l’Homme, 2003.

[40] Dimitrios Meletis. *Graphetik. Form und Materialität von Schrift*. Glückstadt: Verlag Werner Hülsbusch, 2015.

[41] Dimitrios Meletis and Christa Dürscheid. *Writing Systems and Their Use. An Overview of Grapholinguistics*. Vol. 369. Trends in Linguistics Studies and Monographs. De Gruyter, 2022.

[42] *Merriam-Webster's Dictionary of English*. Springfield, MA: Merriam-Webster Inc., 2016.

[43] James Myers. *The Grammar of Chinese Characters. Productive Knowledge of Formal Patterns in an Orthographic System*. Routledge, 2019.

[44] Akira Nakanishi. *Writing Systems of the World*. Clarendon, VT: Tuttle Publishing, 1980.

[45] Heide Oestreich. "Die Erektion im Text". taz, March 7, 2009, https://taz.de/Das-Binnen-I-und-die-taz/!5166721/. 2009.

[46] Manuel Perea. "Revisiting Huey: On the importance of the upper part of words during reading". In: *Psychon. Bull. Rev.* 19 (2012), pp. 1148–1153.

[47] Alexandra Pressigout et al. "Complémentarité entre le bilan neurovisuel orthoptique et l'oculométrie chez des enfants avec des difficultés de lecture". In: *Revue Francophone d'Orthoptie* 15 (2022), pp. 105–109.

[48] Frank Romano. *History of the Linotype Company*. Rochester: RIT Press, 2014.

[49] Rosemary Sassoon. *Handwriting of the Twentieth Century*. London and New York: Routledge, 1999.

[50] Jürgen Scharnhorst. "Die graphische Ebene im Modell des Sprachsystems". In: *Probleme der geschriebenen Sprache. Beiträge zur Schriftlinguistikauf dem XIV. Internationalen Linguistenkongreß 1987 in Berlin*. Ed. by Dieter Nerius and Gerhard Augst. Vol. 173. Linguistische Studien. Berlin: Akademie der Wissenschaften der DDR, Zentralinstitut für Sprachwissenschaft, 1988, pp. 87–102.

[51] Cornelia Schindelin. *Zur Phonetizität chinesischer Schriftzeichen in der Didaktik des Chinesischen als Fremdsprache*. Vol. 13. SinoLinguistica. München: Iudicium, 2007.

[52] Jürgen F. Schopp. In: *Meta Journal des traducteurs Translators' Journal* 53 (2008), pp. 167–183.

[53] Michal Shomer. "Multi-Gender Hebrew: Creating a New Space in the Hebrew Language". In: *Proceedings of Grapholinguistics in the 21st Century, 2022*. Ed. by Yannis Haralambous. Grapholinguistics and Its Applications. Brest: Fluxus Editions, to appear.

[54] Zdenek Smekal et al. "Analysis of neurological disorders based on digital processing of speech and handwritten text". In: *International Symposium on Signals, Circuits and Systems ISSCS2013*. 2013, pp. 1–6.

[55] Erik Spiekermann and E.M. Ginger. *Stop Stealing Sheep and Find Out How Type Works*. Mountain View, CA: Adobe Press, 1993.

[56] Patricia Thaine and Gerald Penn. "Vocalic and Consonantal Grapheme Classification through Spectral Decomposition". In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 367–386.

[57] *The Chicago Manual of Style*. 17th ed. Chicago: The University of Chicago Press, 2017.

[58] Barrie Tullett. *Typewriter Art: A Modern Anthology*. London: Laurence King Publishing, 2014.

[59] Daniel Berkeley Updike. *Printing Types: Their History, Forms, and Use. A Study in Survivals*. Cambridge: Harvard University Press, 1922.

[60] Catherine Warmann. "Installation for the UK Pavilion at Shanghai Expo 2010 by Troika". In: *dezee* (May 2010).

[61] Paul Warren. *Introducing Psycholinguistics*. Cambridge University Press, 2013.

[62] William C. Watt. "Curves as Angles". In: *Writing Systems and Cognition*. Ed. by William C. Watt. Kluwer Academic Publishers, 1994, pp. 215–246.

[63] Noah Webster. *A Compendious Dictionary of the English Language*. New Haven: Hudson & Goodwin, Book-sellers and Increase Cooke & Co. Book-Sellers, 1806.

[64] Suzanne Wehde. *Typographische Kultur. Eine zeichentheoretische und kulturgeschichtliche Studie zur Typographie und ihrer Entwicklung*. Tübingen: Max Niemeyer, 2000.

[65] Bing Xu. *Book from the ground: from point to point*. The MIT Press, 2014.

Chapter 4
Morphemes, Words, Terms

In the previous two chapters, we dealt with the elementary units of speech and writing. When these units are concatenated, either in time (for speech) or in space (for writing), occasionally, meaning emerges. *Morphology* studies the elementary units of meaning, called *morphemes*.

4.1 Words

> Words matter!
>
> The 13th Doctor, *The Haunting of Villa Diodati* S38E8

> Entre la mort et nous il y a des mots.[1]
>
> Jean-Luc André d'Asciano, *Préface à "Efroyabl Ange1" de Iain M. Banks*

One may ask, where morphemes are located. As the reader of this very sentence can testify, they are located in *words*. But what is a word?

Quoting Packard [32, p. 7],

> […] it seems that no matter what the language, we have a hard time providing an exact definition that encompasses all and only those entities that our intuition tells us are words.

Not to mention the fact that the word "word" is polysemic: we all agree that, for example, "cat" is indeed a word, but in the expression "I'll have a word with him" or the French "toucher un mot à quelqu'un"$_{\text{FR}}$, the word "word" has the meaning of discussion or reprimand. In German, the same word "Wort"$_{\text{DE}}$ can carry the meaning of a single word or an utterance. In the first case, the plural will be "Wörter"$_{\text{DE}}$ (as in "Wörterbuch"$_{\text{DE}}$ ⌊dictionary⌋) and in the second "Worte"$_{\text{DE}}$ (as in Mendelssohn's "Lieder ohne Worte"$_{\text{DE}}$ ⌊Songs without Words⌋).

[1] "Between death and us there are words."

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_4

Furthermore, Westerners tend to consider words as either graphemic sequences delimited by interword spaces or punctuation (the *graphemic word*) or as dictionary entries and their inflections (the *lexical word*). Both criteria are incomplete and ambiguous.

On the other hand, some linguists consider that words have not been studied sufficiently in linguistics. For example, Goddard & Wierzbicka:

> The authors of this book believe that it is time to bring words back in and recognize them as part of the core business of linguistics. Systematically exploring the meanings of words must not be relegated to the margins but acknowledged as a vital part of what linguistics is all about and, at the same time, as a necessary path to fully "humanizing" linguistics and reclaiming its place in the humanities, as envisaged centuries ago by Vico. [18, pp. 7–8]

Goddard & Wierzbicka study "universal words," [18, pp. 10–18] that are words that exist in all languages of the world, such as ME, YOU, THING, LIVE, DIE, VERY, etc. These universal words are thoroughly studied in several cultures and languages. Nevertheless, Goddard & Wierzbicka refrain from defining the notion of "word".

4.2 Lexemes

So, how can we appropriately define the notion of *word*? We will not do so. Our somewhat arbitrary starting point will be the notion of *lexeme*: we will consider that a *lexeme* (the elementary unit of the field of *lexicology*) is an entry of some dictionary, relegating the issue of which lexemes should be part of the dictionary to the field of *lexicography*. Lexemes can be inflected (conjugated, declined, etc.), and we will call an inflected version of a lexeme a *word*.

We will start by describing the various classes, also called *parts of speech* (POS), to which lexemes can belong. Then, we will describe how words can be built out of morphemes (inflection, derivation, compounding). Finally, we will consider the question of morpheme/word combinatorics from a different point of view: when does a group of words become the representation of a concept in a specific knowledge domain? Such a group of words is called a *term*, and the field studying terms is called *terminology*.

4.3 Parts of Speech

he main *lexeme classes*, also called *parts of speech*, were described as early as in the 2nd century BC by Dionysius Thrax. He describes nouns, verbs, participles, articles, pronouns, prepositions, adverbs, and conjunctions. One would expect Dionysius's definition of a "noun" to be very philosophical. It is not: for him, a noun is a part of speech that can be *declined* ("πτωτικόν"$_{\text{EL}}$) and that denotes a *concrete* or an *abstract entity*. And a "verb" is a word that *cannot be declined* ("ἄπτωτον"$_{\text{EL}}$), but that has *tenses*, *numbers*, and *persons* and denotes an *action acted* ("ἐνέργεια"$_{\text{EL}}$) or an *action* *inflicted* (“πάθος”$_{\text{EL}}$). Reading Dionysius, one has the feeling that, in 200 BC already, he was not discovering anything but was merely writing down common knowledge. Nowadays, 22 centuries later, according to Common Core State Standards,[2] five main parts of speech are part of Foundational Skills and taught to third-graders: nouns, pronouns, verbs, adjectives, and adverbs—among the five, only one (namely adjectives) was not already included in Dionysius’s text, 200 BC.

The international annotation framework “Universal Dependencies” (UD) has provided a list of “universal part-of-speech tags,” universal in the sense that not every language uses all types but that any word in any language can be mapped to one of these types. We display them in Table 4.1.

Table 4.1 Universal part-of-speech tags (UPOS), taken from [28]

Traditional POS	UPOS	Category
noun	NOUN	common noun
	PROPN	proper noun
verb	VERB	main verb
	AUX	auxiliary verb or other tense, aspect, or mood particle
adjective	ADJ	adjective
	DET	determiner (including article)
	NUM	numeral (cardinal)
adverb	ADV	adverb
pronoun	PRON	pronoun
preposition	ADP	adposition (preposition/postposition)
conjunction	CCONJ	coordinating conjunction
	SCONJ	subordinating conjunction
interjection	INTJ	interjection
–	PART	particle (special single word markers in some languages)
–	X	other (e.g., words in foreign language expressions)
–	SYM	non-punctuation symbol (e.g., a hash (#) or an emoji)
–	PUNCT	punctuation

Nouns can be *proper* (referring to unique entities, such as “London,” “Clara,” or “The Doctor”) or *common* (referring to classes of entities, such as “town,” “person” or “idea”). They can be *countable* (“man,” “box,” “word”) or *mass nouns* (“butter,” “furniture,” etc.), a special case of countable noun being the *collective* noun, which refers to a group and is sometimes considered as being singular and sometimes plural (“the government was…” or “the government were…” are both acceptable).

Pronouns are words that stand for nouns or noun phrases. They can be *demonstrative* (“that,” as in “that’s where I live”), *indefinite* (“some”), *interrogative* (“which”), *negative* (“none”), *possessive* (“mine”), *“relative”* (“that,” as in “the house that I live in”), etc. French uses *personal* pronouns extensively, as in “je le leur dis”$_{\text{FR}}$ ⌊I’m telling it to them⌋, where the subject, direct object, and indirect object are all pronouns.

[2] https://learning.ccsso.org/wp-content/uploads/2022/11/ELA_Standards1.pdf

Adjectives can be *prepositive* (“a lovely house”) or *postpositive* (“rooms available”). In French, many adjectives can be used in both ways, with meanings that may differ (as in “un seul ami”$_{\text{FR}}$ ⌊only one friend⌋ ≠ “un ami seul”$_{\text{FR}}$ ⌊a lonely friend⌋). Adjectives can also be *nominalized*, i.e., used as nouns, like in “These are the best.”

Adverbs are a catch-all class of word modifiers that are neither declined nor conjugated but have all kinds of other inflections, such as degree (“loudly,” “loudlier,” “loudliest”), etc. In English, they usually modify verbs (“I drink slowly”), adjectives (“very young”), other adverbs (“very very young”), prepositions (“probably at home”), etc. In French, they can also modify nouns and even proper nouns, as in “ça c’est très Michel”$_{\text{FR}}$ ⌊this is very much in the manner of Michel⌋.

Sometimes, a lexeme can belong to several classes, such as “fish” (noun, but also verb “to fish”). Shifting from one class to another is called *conversion*. For some linguists, conversion occurs only when the lexemes in the two classes are strictly identical (such as the French “parler”$_{\text{FR}}$, which can be the infinitive of the verb “to speak” or a noun meaning “parlance”). Others consider that conversion can also be obtained by adding a suffix, as in inflection (e.g., the German noun “Liebe”$_{\text{DE}}$ ⌊love⌋ converted into the verb “to love” by adding a <n> suffix: “lieben”$_{\text{DE}}$). The best-known example of conversion is one of the words “buffalo” (derived from the Greek “βούβαλος”$_{\text{EL}}$) in the sentence

Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo [10],

that can be interpreted as “the buffaloes from Buffalo that are bullied by buffaloes from Buffalo bully other buffaloes from Buffalo.” The lowercase word “buffalo” acts thrice as a noun and once as a verb, while the named entity “Buffalo” acts as a noun adjunct.

4.4 Morphemes

As mentioned, meaning emerges when elementary phonological or graphemic units are concatenated in a language-specific way. Minimal units of meaning (i.e., such that if we remove a unit, the meaning either changes or vanishes) are called *morphemes*.

For example, when concatenating the phonemes /k/, /æ/ and /t/ or the graphemes <c>, <a> and <t>, the meaning of “cat” arises, therefore /kæt/ and <cat> are (two representations of) a morpheme.

When we wish to refer to more than one cat, we add a phoneme /s/ or a grapheme <s>. We can consider this phoneme or grapheme to have the abstract meaning of +PLURAL. Therefore, it is also a morpheme, but of a particular kind since it can be used only in conjunction with other morphemes. We call the oral /s/ of /kæts/ or the written <s> of <cats> a *bounded* or *grammatical* morpheme. All morphemes that can be used in isolation are called *free* morphemes.

A language can have an arbitrary and increasingly growing number of free morphemes. According to its position vs. free morphemes, a bounded morpheme is called a *suffix* (after the free morpheme, as “s” in “cats”), a *prefix* (as “ge”$_{\text{DE}}$ in the

German “gesehen”$_{\text{DE}}$ ⌊seen⌋), an *infix* (inside the free morpheme)—in the more general case, it is called an *affix* (see for example §4.8.1). These phenomena should not be confused with the one of *interfix*, which is a purely phonetic infix, as the “o” in “speedometer,” which carries no meaning whatsoever.

In the following, we will consider the three most important morphological phenomena, namely *inflection*, *derivation* and *compounding*.

4.5 Inflection

Inflection is an important phenomenon studied by morphology. It consists in combining a free morpheme with bounded morphemes to change the properties of number, gender, definiteness, case, person, mood, voice, aspect, tense, and degree. It is called *declension* in the case of nouns and adjectives and *conjugation* in the case of verbs. We consider that the forms we obtain by inflecting a given noun, adjective, adverb, or verb belong to the same *lexeme* (e.g., “go,” “goes,” “went,” “going” belong to the same lexeme “(to) go”).

Inflection can be pretty different among languages. In particular, each language has its combination of numbers, genders, and cases:

Language	numbers	genders	cases
English	2	0	5
French	2	2	0
German	2	3	4
(Ancient) Greek	3	3	5
Russian	3	3	6
Arabic	3	2	3
Finish	2	0	14
Japanese	0	0	8

Absence of one of these features in a language does not imply that it is poorer, but rather that the corresponding functions are achieved by other means than by bounded morphemes. For example, in Russian, using a grammatical case allows one to distinguish a subject from an object. For example, sentences “мальчик съел вишню”$_{\text{RU}}$ and “вишню съел мальчик”$_{\text{RU}}$ (where we have permuted the first and the third word) both mean “the boy ate the cherry” because of the empty bounded morpheme of “мальчик”$_{\text{RU}}$ denoting nominative case and the bounded “ю”$_{\text{RU}}$ morpheme in “вишню”$_{\text{RU}}$ denoting accusative. In French and other fixed-word order language, it is the order ☞ 95
of words that carries this kind of information: in declarative sentences, the subject is placed before the verb, and the object after it, and no permutation is possible without disrupting the grammaticality of the sentence.

Let us briefly describe the ten inflectional categories:

- *number* expresses count distinctions: most languages either have no bounded morpheme for number (e.g., in Japanese, “本”$_{\text{JA}}$ stands both for “book” and “books”) or mark singular and plural (“cat” vs. “cats,” in which case “s” is the morpheme of the plural). Some languages, such as Arabic and ancient Greek, also use a *dual* when denoting two items (for example, in "ἄνθρωπος"$_{EL}$ ⌊man⌋, "-ος"$_{EL}$ is the morpheme of nominative singular, it becomes "-οι"$_{EL}$, in the plural, and "-ω"$_{EL}$ in the dual). Russian uses different morphemes for one item, for two to four items, and for five or more items (for example, in "книга"$_{RU}$ ⌊book⌋, "-а"$_{RU}$ is the morpheme of the nominative singular, it becomes "-и"$_{RU}$, for less than 4 items, and by the empty morpheme for five or more items: "книг"$_{RU}$);

- *gender*: many languages have a grammatical gender system with the standard divisions masculine/feminine (e.g., in the Italian "bello"$_{IT}$ ⌊beautiful$_{♂}$⌋ "-o"$_{IT}$ is the morpheme of the masculine and "-a"$_{IT}$ the morpheme of the feminine, as in "bella"$_{IT}$), or masculine/feminine/neuter (e.g., in the German "großer"$_{DE}$ ⌊big$_{♂}$⌋, "-er"$_{DE}$ is the morpheme of the masculine, "-e"$_{DE}$ of the feminine and "-es"$_{DE}$ of the neuter). Grammatical gender can often be unrelated to the meaning of the word (e.g., in French, Italian, Spanish, and Greek, the sun is of masculine grammatical gender, and the moon is of the feminine gender, while in German, it is the opposite), it may even change between singular and plural (in French, the word "amour"$_{FR}$ ⌊love⌋ is masculine in the singular and feminine in the plural) and some words may have both genders (in French: "après-midi"$_{FR}$ ⌊afternoon⌋). Gender morphemes are used in nouns and adjectives, but in some languages, as in Arabic, also in verbs;
- *definiteness*: a distinction between referents to an entity identified in the context or to an unidentified entity (as in the English definite article "the" and indefinite article "a"). In Arabic, indefiniteness is marked by a suffix /n/, graphically represented by a duplication of the short vowel grapheme of the last syllable, e.g., "كتابُ"$_{AR}$ ⌊the book⌋ vs. "كتابٌ"$_{AR}$ ⌊a book⌋;
- *case* expresses grammatical functions for a noun and its modifiers (such as adjectives). Common cases are nominative (subject), genitive (possessor), dative (indirect object), accusative (direct object), vocative (addressee), ablative (movement away from), prepositional (with prepositions) and instrumental (means or tool). Examples: in the Russian "кукла"$_{RU}$ ⌊doll⌋, "-а"$_{RU}$ is the morpheme of the nominative, "-ы"$_{RU}$ of the genitive, "-е"$_{RU}$ of the dative, "-у"$_{RU}$ of the accusative, "-е"$_{RU}$ of the prepositional, "-ой"$_{RU}$ of the instrumental; in the Greek "ἄνθρωπε"$_{EL}$ ⌊human⌋, "-ε"$_{EL}$ is the morpheme of the vocative; in the Finnish "talolta"$_{FI}$ ⌊house⌋, "-lta"$_{FI}$ is the morpheme of the ablative;
- *person*: the distinction between speaker (1st person), addressee (2nd person) and others (3rd person). In the German "(Ich) spreche"$_{DE}$ ⌊I speak⌋, the 1st person morpheme is "-eche"$_{DE}$), in the 2nd person the form becomes "Du sprichst"$_{DE}$) and in the 3rd person "Sie spricht"$_{DE}$;
- *mood* expresses the speaker's attitude toward what she says. Common moods are indicative (factual, as in the French "il a"$_{FR}$ ⌊he has⌋), subjunctive (hypothetical, "(il) aie"$_{FR}$), conditional (dependent upon conditions "(il) aurait"$_{FR}$), imperative (command, "aie"$_{FR}$), optative (wish), jussive (command to an absent agent), etc. An example of optative mood is the Greek "λύοις"$_{EL}$ ⌊(I hope that you) solve⌋, in which case "λύεις"$_{EL}$ would be an indicative and "λύῃς"$_{EL}$ a subjunctive. An example of jussive mood is the Finnish "eläköön"$_{FI}$ ⌊let him/her/it live!⌋ where

“-köön”$_{\text{FI}}$ is the morpheme of the jussive mood—the morpheme of the imperative mood would be “-ä”$_{\text{FI}}$ as in “elää”$_{\text{FI}}$ ⌊live!⌋;
- *voice* expresses the relationship between a verb and the agents identified by its subject and object. Common voices are: active (subject performs action), passive (subject is recipient of action), and middle (subject performs an action and is also affected by that action). An example in Ancient Greek: active voice “λύσω”$_{\text{EL}}$ ⌊I will untie⌋, middle voice “λύσομαι”$_{\text{EL}}$ ⌊I will untie myself⌋, passive voice “λυθήσομαι”$_{\text{EL}}$ ⌊I will get untied (by somebody else)⌋;
- *aspect* expresses how an action extends over time: actions can refer to an individual event (perfective aspect, e.g., “I have eaten”) or to an action that exists continuously or repetitively (imperfective aspect, “I eat”). Furthermore, aspect can be progressive (“I am eating”) or not (“I eat”);
- *tense* expresses time reference. Most languages have basic tenses (past, present, and future, e.g., the French “(je) mange”$_{\text{FR}}$ ⌊I eat⌋, “(je) mangeais”$_{\text{FR}}$ ⌊I was eating⌋, “(je) mangerai”$_{\text{FR}}$ ⌊I will eat⌋) but there are also less common tenses such as hodiernal (today), crastinal[3] (tomorrow) and hesternal (yesterday) tenses, as well as relative tenses: pluperfect (the past of the past, e.g., “(j’)avais mangé”$_{\text{FR}}$ ⌊I had eaten⌋), future perfect (the past of the future, “(j’)aurai mangé”$_{\text{FR}}$ ⌊I will have eaten⌋), posterior (the future of the past), etc.;
- *comparison* or *degree* is used for building comparative and superlative adjectives and adverbs (as in “small,” “smaller,” “smallest”).

Also, in Arabic, there is a feature of nouns that is not represented by a standard affix ☞ 74 but implies specific affixes for the plural and specific agreements with other parts of the phrase. This feature is *humanness* [37, p. 125] (or *animacy* [28, p. 263]) and it determines whether the adjective of a plural noun is in plural form (for human referents) or in singular feminine form (for nonhuman referents), compare “الجدد المعلمين”$_{\text{AR}}$ ⌊the new teachers⌋, in which جدد is the plural form of the adjective “جديد”$_{\text{AR}}$ ⌊new⌋, and “البيوت الجديدة”$_{\text{AR}}$ ⌊the new houses⌋, where جديدة is the feminine singular form of the same adjective.

Finally, other features, taken into account by the Universal Dependencies annotation framework are *evidentiality* (to distinguish between what one has felt, or one has circumstantial evidence for, visual evidence, etc., in Amerindian or Slavic languages), *polarity* (positive or negative), *politeness* (elevated, formal, humble and informal, mainly in Japanese and Korean) and *clusivity* (the distinction between inclusive “we” and exclusive “we”).

4.6 Derivation

Contrarily to inflection, when we *derive* a free morpheme by adding bounded morphemes, we create new lexemes: for example, “hard” is a lexeme, “hardness” (the property of being hard) is a different lexeme, the meaning of which can be compo-

---

[3] From which the word for many people’s favorite activity, procrastination, has been derived.

sitionally derived from “hard” and “-ness.” But the semantics of derivation is not always easily predictable, e.g., there is “hardly,” the meaning of which cannot be predicted from the morphemes “hard” and “-ly,” while such a prediction is possible in the case of “slowly.”

The most common derivational mechanisms are:

- *suffixation*: adding a bounded morpheme as a suffix, as in “type” → “typical,” “typicality,” “typology,” “typological,” “typing,” “typo,” etc.;
- *prefixation*: adding a bounded morpheme as a prefix, as in “cover” → “recover,” “discover,” “uncover,” “undercover,” “softcover,” “bedcover,” etc.;
- *abbreviation*: a sequence of the initial graphemes of a group of words, it may be
  - an *acronym*, when the sequence is pronounced as if it were a word, e.g., “UNESCO,” “radar,” “Tardis,” “UEFA”;
  - or an *initialism*, when the grapheme names are spelled, as in “USA,” the German “GmbH”$_{\text{DE}}$ that illustrates the use of bicamerality in German (that is: the fact that nouns are capitalized so that the “m” cannot be the initial of a noun) or the French “SNCF”$_{\text{FR}}$;

  It is also possible to have hybrids, i.e., partly acronyms and partly initialisms, as in “JPEG” (pronounced /ˈʤeɪpɛg/);
- *clipping* removes part of a morpheme. It can be
  - *apheresis*, when the initial part is removed (as in “bot” for “robot,” “gator” for “alligator,” etc.);
  - *syncope*, when a medial part is removed (as in the Latin “parabolare”$_{\text{LA}}$ that became “parlare”$_{\text{IT}}$ in Italian);
  - *apocope*, when the final part is removed (as in “gym” for “gymnastics,” or the French “ciné”$_{\text{FR}}$ or “cinéma”$_{\text{FR}}$ for “cinématographe”$_{\text{FR}}$).

  In Greek, there is a graphemic syncope convention of truncating word interiors and adding a slash: “ἀνελ/ρας”$_{\text{EL}}$ for “ἀνελκυστήρας”$_{\text{EL}}$ ⌊elevator⌋, “Θεσ/νίκη”$_{\text{EL}}$ for “Θεσσαλονίκη”$_{\text{EL}}$ ⌊Thessaloniki⌋, etc.;
- *reduplication*, when a morpheme is repeated, as in “bye-bye,” “fifty-fifty,” etc. Sometimes the duplicated morpheme bears a small change, as in “boogie-woogie” or in the famous Whovian idiom “timey-wimey,” used by the 10th Doctor in the episode *Blink* S29E10, probably inspired by the reduplication “teenie-weenie” from the song *Itsy Bitsy Teenie Weenie Yellow Polkadot Bikini* (1960). A particular case of reduplication is the Yiddish *shm-reduplication* as in “breakfast-shmeakfast,” connoting doubt or irony [29].
  In French, sometimes only the first syllable is reduplicated: “fille”$_{\text{FR}}$ ⌊girl⌋ → “fifille”$_{\text{FR}}$ ⌊girl⌋ (hypocoristically), “père”$_{\text{FR}}$ ⌊father⌋ → “pépère”$_{\text{FR}}$ affectionate name for a male, etc.
  In Greek, reduplication denotes a higher degree, as in “πρῶτα”$_{\text{EL}}$ ⌊first⌋ (adv.) → “πρῶτα-πρῶτα”$_{\text{EL}}$ ⌊first of all⌋, “πάνω”$_{\text{EL}}$ ⌊on top of⌋ → “πάνω-πάνω”$_{\text{EL}}$ ⌊on the highest level⌋, or progression, as in “ἕνα”$_{\text{EL}}$ ⌊one⌋ → “ἕνα-ἕνα”$_{\text{EL}}$ ⌊on-by-one⌋, “λίγο”$_{\text{EL}}$ ⌊a little⌋ → “λίγο-λίγο”$_{\text{EL}}$ ⌊in small quantities⌋.

In Indonesian, reduplication has many functions, the most common being the plural number: “anak”$_{\text{ID}}$ ⌊child⌋ vs. “anak-anak”$_{\text{ID}}$ ⌊children⌋ [33].

4.7 Compounding

Compounding combines two or more lexemes, called *components*. Typical examples are “brainstorm,” or the German “Milchversorgung”$_{\text{DE}}$ ⌊milk distribution⌋, or the French “savoir-vivre”$_{\text{FR}}$. Sometimes, a phoneme/grapheme is inserted between components for phonetical reasons, like in “speedometer,” this phenomenon is called *infixation*. The forms of the lexemes can be inflected, as in the German “Gebrauchsanweisung”$_{\text{DE}}$ ⌊user manual⌋ where the first component “Gebrauchs”$_{\text{DE}}$ contains the “-s”$_{\text{DE}}$ morpheme that marks the genitive case.

The difference between compounds and derivations is that the former are built out of lexemes only, while the latter are built out of a lexeme and one or more grammatical affixes. Booij [9, p. 30] remarks that there are notable exceptions to this. First of all, there exist compounds one or more components of which are neither lexemes nor grammatical morphemes, such as in the word “cranberry,” where “cran” is not an English morpheme (it has some etymological connection to “crane” but the meaning of “crane” is irrelevant to cranberries). These are called *cranberry compounds*.

An essential case of compounds is the one of *neoclassical compounds*, built from Greek and Latin morphemes, such as “geology,” “xenophobia,” “transform,” “interaction,” etc. Often, they contain only Greek morphemes (as in “ecology”) or only Latin morphemes (as in “conscience”). Still, hybrids are also possible (as in “television,” which is Greek-Latin, or “interphone,” which is Latin-Greek). These morphemes can also combine with ordinary English lexemes, as in “ecofriendly” or “biohackers.” The question raised by [9] is whether “geo-” should be considered a lexeme (which it is in Greek: “γῆ”$_{\text{EL}}$ ⌊Earth⌋) or an affix, that is, a grammatical morpheme. The former cannot be true since no English lexeme “geo” exists (even though a well-known magazine carries that name). If the latter were true, it would mean that dozens (hundreds?) of Greek and Latin morphemes would automatically become grammatical morphemes of English, not to mention the fact that by combining these “grammatical” morphemes, as in “biology,” we would have lexemes without any free morpheme, which is absurd. (See Exercise 3-3, p. 86 on neoclassical compounds.)

By having their status, neoclassical compounds live in a grey area between affixes and lexemes—therefore, they are called *affixoids* [9, p. 86]. Besides neoclassical compounds, one occasionally encounters other affixoids, i.e., lexemes that behave like affixes, in the sense that they can be combined with an arbitrary number of other lexemes. Ruppenhofer [36] mentions the example of “Lieblings-”$_{\text{DE}}$ ⌊favorite⌋ in German, as in “Lieblingsgericht”$_{\text{DE}}$ ⌊favorite dish⌋, etc.

An interesting word creation method is the one of *portmanteau* lexemes (from the French word “porte-manteau”$_{\text{FR}}$ ⌊coat rack⌋, which is a portmanteau itself[4]) produced not by a concatenation but by a *merge* of free morphemes (in the sense that at least one of the two morphemes is truncated), such as “smog” (from “smoke” and “fog”) or “Brexit” (form “Britain” and “exit”) or the French “franglais”$_{\text{FR}}$ ⌊Frenglish⌋ (from “français”$_{\text{FR}}$ and “anglais”$_{\text{FR}}$).

And finally, to emphasize a given polysyllabic word, one can insert an expletive such as “bloody” inside the lexeme, as in “un-bloody-believable” (the title of a COVID-era comedy in New Zealand). This phenomenon is called *tmesis* (from “τμῆσις”$_{\text{EL}}$ ⌊division⌋) and is also observed in words such as “whatsoever” [23], where “so” is placed inside “whatever.”

4.8 Astonishing Morphologies: Semitic Languages and Lojban

4.8.1 Semitic Languages

Ranking languages of the world by the total number of speakers (L1 and L2), Modern Standard Arabic (MSA) is ranked fifth, ex æquo with French (both count around 274 million speakers each, according to the 2022 edition of *Ethnologue*[5]). Other Semitic languages, such as Amharic, Hebrew, Syriac, and Maltese, have fewer speakers (around 20 million for Amharic, 9 million for Modern Hebrew, 1 million for Syriac, and 552 thousand for Maltese). They all share morphological properties based on the combinatorics of two structures: *roots* and *patterns* (also called *templates*) (see Ryding [37, pp. 45–49] for Arabic and Glinert [17, pp. 458–459] for Hebrew).

Roots are sequences of 2, 3 or 4 consonants R_1R_2, $R_1R_2R_3$, $R_1R_2R_3R_4$ (cases of 5 consonants are rare) subject to some phonological constraints, e.g., “R_1 cannot be equal to or have the same place articulation as R_2,” “R_2 and R_3 can be identical but, if not identical, cannot have the same place of articulation,” etc. (see Spagnol [40, pp. 20–25] for more details).

Roots refer to semantic domains, for example ك ت ب (Hebrew כ ת ב) k-t-b refers to the domain of “writing,” ف ع ل (פ ע ל) f/p-ʕ-l to the domain of “activity,” ق ر ب (ק ר ב) q-r-b to the domain of “proximity”. There are around 5,000 roots and new roots can be created using standard morphological methods, such as clipping (as in Arabic root س م ع ل s-m-ʕ-l ⌊to say salam-aleikoum⌋), acronymization (as in Hebrew root ת נ כ t-n-kh ⌊Tanakh⌋, an acronym the expansion of which is “תורה נביאים וכתובים”$_{\text{HE}}$ ⌊Tora Prophets (and) Scriptures⌋), or compounding (as in Hebrew ר מ ק ל r-m-q-l used in the word “רמקול”$_{\text{HE}}$ ⌊loudspeaker⌋, from “רם”$_{\text{HE}}$ ⌊loud⌋

[4] Words that are themselves what they describe are called *autological*, like “pentasyllabic” (which has five syllables), “readable,” “unhyphenated,” “prefixed,” “vowelled,” “written,” etc. The word “portmanteau” obeys the truncation rule of the first component, which has lost its final “-e” compared to the French original.

[5] https://www.ethnologue.com/

and "קול"$_{\text{HE}}$ ⌊sound⌋) [15]. Sometimes convenient loanwords are used to create new roots, such as ف م ل f-l-m from the English word "film," which is written "فيلم"$_{\text{AR}}$ /film/ with the plural "أفلام"$_{\text{AR}}$ /ʔaflām/.

Patterns are grammatical morphemes that are intertwined with roots: for a root consisting of 2 (resp. 3 or 4) consonants, R_i, we will have patterns of 3 (resp. 4 or 5) elements T_i so that we can build words

$$T_0R_1T_1R_2T_2, \qquad T_0R_1T_1R_2T_2R_3T_3, \qquad T_0R_1T_1R_2T_2R_3T_3R_4T_4.$$

Using this notation, the T_is can be empty or consist of one or more vowels or consonants (mostly vowels [37, p. 48]). Patterns serve to build *all* inflected and derived forms. Concerning derivation, for Arabic verbs, there exist fifteen derivation processes called *forms* (or *measures* or *derived conjugations*) [37, pp. 433–598], out of which ten are common and five obsolescent. Hebrew has a similar system of seven forms [17, pp. 460–461].

The combinatorial potential of roots and patterns is very high, so millions of forms can be created theoretically. In Table 4.2, the reader can see a few examples based on the root ق ر ب q-r-b (for "proximity").

Table 4.2 Examples of Arabic inflected and/or derived forms

R_1aR_2uR_3a	**q**a**r**u**b**a	قَرُبَ *being close*	Verb (Form I)
R_1aR_2R_2aR_3a	**q**a**rr**a**b**a	قَرَّبَ *bring closer*	Verb (Form II)
maR_1R_2uR_3a(t)	ma**qr**u**b**a(t)	مَقْرُبَة *proximity*	Noun
R_1aR_2īR_3u	**q**a**r**ī**b**	قَرِيبُ *relative, parent*	Adjective
taR_1R_2īR_3iyy	ta**qr**ī**b**iyy	تَقْرِيبِي *approximate*	Adjective
taR_1āR_2uR_3u	ta**q**ā**r**u**b**u	تَقارُبُ *bringing closer*	Verb (Form VI)
ʔiR_1taR_2aR_3a	i**q**ta**r**a**b**a	إقْتَرَبَ *getting closer*	Verb (Form VIII)
R_1uR_2R_3a	**q**u**rb**a	قُرْبَ *close to*	Adverb

4.8.2 Lojban

> Loglan is fine for syllogism, circuitry, and mathematical calculations but lacks flavor. Useless for gossip or to whisper into girl's ear.
>
> Robert A. Heinlein, *The Moon is a Harsh Mistress*

In the previous section, we considered Semitic languages, with more than 300 million speakers. Lojban is a language spoken and/or written only by a few hundred people (as of October 2023, the "Lojban Beginners" online forum had 727 members). It is a strictly logical language, the development of which started in 1987. A tax-exempt scientific-educational organization maintains it, the Logical Language Group, the office of which is located in Fairfax, Virginia.

It all started with a paper by Brown published in 1960 in the *Scientific American* [11], describing an ancestor of Lojban, called *Loglan*. This article inspired Robert Heinlein, who used Loglan as a basic ingredient of his 1966 book *The Moon is a Harsh Mistress*. Lojban forked from Loglan in 1987.

Lojban's morphology is extremely simple and straightforward: there are three parts of speech: grammatical words (“cmavo”$_{\text{LOJ}}$), content words (“brivla”$_{\text{LOJ}}$), and proper nouns (“cmene”$_{\text{LOJ}}$).

Cmavo play the role of determiners, conjunctions, prepositions, numbers, and even punctuation marks. And they go even further: attitudes, such as desire, despair, hesitation, surprise, etc., usually conveyed by prosody in speech, are explicitly introduced by cmavo words, called *attitudinals*. This is the whole idea of “logical language,” namely to strive towards total transparency between oral and written modalities.

Brivla are content words, playing the roles of nouns, verbs, adjectives and adverbs. There are three kinds of brivla: the *primitive roots* (“gismu”$_{\text{LOJ}}$) from which all other brivla are built, compounds (“lujvo”$_{\text{LOJ}}$) and loanwords from other languages (“fu'ivla”$_{\text{LOJ}}$). Cowan describes how gismu words have been chosen:

> Almost all Lojban gismu are constructed from pieces of words drawn from other languages, specifically Chinese, English, Hindi, Spanish, Russian, and Arabic, the six most widely spoken natural languages. For a given concept, words in the six languages representing that concept were written in Lojban phonetics. Then, a gismu was selected to maximize the recognizability of the Lojban word for speakers of the six languages by weighting the inclusion of the sounds drawn from each language by the number of speakers. [12, p. 53]

What is not said in this paragraph is that because of the distribution of languages among the world's population, most gismu words stem from English and Chinese. Loanwords are phonetically adjusted and prefixed so that “spaghetti” becomes “cidj,r,spageti”$_{\text{LOJ}}$, where the prefix “cidj,-”$_{\text{LOJ}}$ stands for “food” and “,r-” plays the role of a hyphen.

Not only does Lojban avoid inflection, but the three parts of speech are immediately recognizable by their phonemic/graphemic patterns. For example, a brivla always ends in a vowel and contains a consonant pair in the first five letters.

Last but not least, the morphology and syntax of Lojban can be represented by a ☞ 249 regular grammar, included in the form of Lex and Yacc code in [12].

All Lojban utterances are perfectly logical, and no room is left for ambiguity. Sentences are obligatorily separated by “.i”$_{\text{LOJ}}$, and there are mandatory constituent end-delimiters (“ku”$_{\text{LOJ}}$ after every noun phrase, “kei”$_{\text{LOJ}}$ after every embedded clause). According to Nicholas, the presence of these delimiters makes punctuation unnecessary:

> You can put exclamation marks and commas before or after the spoken delimiters and attitudinals, but the spoken delimiters and attitudinals are meant to be the on-the-record, linguistically relevant expressions of that type of meaning; the rest is not meant to be determinative and is treated as decorative. The meaning is considered to be the same, whether you put the punctuation in or not or use intonation. And if the meaning intended is different from the

> plaintext you've produced, then that's considered to be you using Lojban erroneously, not an inadequacy of the language. [30]

Lojban is a characteristic example of a language that aims to be "ideal" (see also Eco [14] and Landragin [25, pp. 120–122]). It may never become as widespread as English or Chinese (or even Esperanto). Still, it is a valuable initiative and deserves all the support it can get. (See also Exercises 3-4, 4-3 and 5-3, on p. 86, 131 and 192, on the morphology, syntax and semantics of Lojban.)

4.9 Terms and Collocations

> Tea, but the strong stuff. Leave the bag in.
>
> The Doctor (in a saloon) *A Town Called Mercy* S33E3

Terms are located at the interface between morphology and syntax. They are sequences of morphemes that may be represented by a single word or by multiple words. What distinguishes them is not their form (which usually belongs to a small number of patterns) but their function: they are *the elementary building blocks of a knowledge domain*. Typical terms are "toxoplasmosis" (one word, three morphemes) or "undirected graph" (two words, four morphemes), etc. Terms describing the same concept is multiple languages can have different forms, as in "group order" (two words, noun-noun pattern), which in French is "ordre de groupe"$_{\text{FR}}$ (three words, noun-preposition-noun pattern), and in German "Gruppenordnung"$_{\text{DE}}$ (one word, noun-noun compound).

Terms should not be confused with *collocations*. A collocation is a sequence of words that happen to occur frequently together. A paradigmatic example of collocation is the one of "strong tea" vs. "powerful tea" [39, 27]. In Google, "strong tea" (with quotes) has (as of October 2023) 755 thousand hits, while "powerful tea" has only 53 thousand hits, many of them probably written by non-native English speakers[6]. As a tea producer's Web site states, "A "strong tea" refers to the state of a tea, not the entire tea category," in other words: "strong tea" is not a term in the field of trade and is not widely used in marketing. The author has never seen it on a restaurant's menu either, so it is not a term in the knowledge domain of gastronomy, contrarily to "green tea," which is the name of a family of teas related to a specific plant, the *Camelia sinensis*—"green tea" is clearly a term, both in the field of trade as in the field of gastronomy.

Morphology is relevant in term detection and extraction, as they are formally represented by specific part-of-speech patterns having specific frequency properties. For example, a word pattern ABC has a high chance of being a term if it is a collocation and if it is significantly more frequent than $A'BC$ or $AB'C$ or ABC'

[6] And some pages, such as Leo Selivan's blog *Powerful tea and other (im)possible collocations* `http://leoxicon.blogspot.com/2017/05/powerful-tea-and-other-impossible.html`, which refers to the linguistic expression and not to tea…

(for any A', B' and C', synonyms of A, B and C), and significantly less frequent than any $DABC$ or $ADBC$ or $ABDC$ or $ABCD$ (for any word D). For example, "floating point number" is a term because it is a collocation ("floating point" is more frequent than "floating banana"). If we replace any one of the words with a synonym, we get less frequent expressions (e.g., "drifting point number," "floating dot number," "floating point quantity"). When we add an additional word (such as "large floating point number" or "floating point number techniques"), we get, once again, lesser frequent expressions. Frantzi & Ananiadou [16] define the notion of *termhood* as a numeric value representing the probability of a word sequence being a term. They give a formula to calculate it based on the frequencies of super- and subsequences.

4.10 Psycholinguistic Aspects

> ALL MIMSY WERE THE BOROGOVES
>
> The meanings of the words are as follows:
> *Mimsy* (whence *Mimserable* and *Miserable*): "Unhappy."
> *Borogove*: An extinct kind of Parrot. They had no wings, beaks turned up, and made their nests under sun-dials: lived on veal.
>
> Lewis Carroll, *Stanza of Anglo-Saxon Poetry*

We "know" around 20,000 words in our native language. Still, we realize instantly that the word "borogove" does not exist, at least it didn't exist before Lewis Carroll invented it in 1855, and Lewis Padgett used it in his SF story *Mimsy were the borogoves* in 1943. Now it attains 20 thousand Google hits, so in a sense, it *does* exist.

As first made explicit by Bloomfield [7, 3], we store words in a *mental lexicon*. Here, "lexicon" means some means of "finding" words almost instantly—"finding" in the sense of checking whether a given word is part of the language and accessing its meaning quickly. The mental lexicon also allows the building of new words from existing ones using morphological rules.

Finding Words

Finding words (*lexicalization*) is a two-step process: we start with an abstract concept and retrieve an abstract form (equivalent to the notion of *lemma*), and afterward, we construct the word form. These two processes are called *lexical selection* and *phonological encoding*.

The reader may ask: "How do we know what happens in our minds in the first place?" Well, evidence about the two steps of lexicalization is provided by *speech errors* (or *slips of the tongue*). Collecting speech errors is a fancy hobby and a very serious way of getting insight into the workings of the mind. Harley [21] mentions the following *substitution*:

> Utterance: I've *eaten all my library books. (Target: I've read all my library books. Context: The speaker reported that he was hungry and was thinking of getting something to eat.)

This error shows that there has been confusion between the lemmas "to read" and "to eat," but *after* the confusion, the speaker conjugated the verb to obtain its present perfect tense form. Consequently, inflection must be performed after lexical selection, and therefore, logically, mental lexicon entries must be uninflected. But English has a rather simple verb morphology (compared to French, Greek, or Finnish), so one may question: what if there were separate mental lexicon entries for each tense? Studies in other languages with complex morphologies (like Arnaud [2], for French) have shown that lexical selection is indeed uninflected, except for irregular forms.

Besides substitution, another type of common speech error is *blends*:

> I collected my hand *buggage

where "buggage" is a blend between "baggage" and "luggage," or the famous Bushian quote [41]:

> They *misunderestimated me

which is a blend between "misunderstanding" and "underestimating"[7]. Blends most often occur between synonyms or near-synonyms ("stomach" + "tummy" → "stummy," etc.), which means that when we search for a word to represent a given context, we get access to all synonyms and near-synonyms, so that such errors get possible. Accessing entries in the mental lexicon is called *activation*.

One of the oldest psychological test methods is the one of *word associations* ("What word does X bring to your mind?"). Word association induces activation through associative links between lemmas, which can lead us far away from the initial concept. Lemmas obtained by associative links do not occur in blends but in substitutions, as in

> tuck your *trousers into your boots

where "shoes" has been substituted by "trousers," the word association comes from the fact that both are clothing items.

Three more factors for word association are frequency (how frequent a word is), *concreteness* (how concrete is a concept), and *imageability* (can we easily form a mental image of the concept?) [34]. Predictability is also taken into account (the word "broth" is predictable in the idiomatic expression "too many cooks spoil the …").

Finally, associations can also be based on phonetic similarities, in particular concerning the beginning and end of words (the so-called *bathtub effect* [1]). The character Mrs. Malaprop (from the French idiom "mal à propos"$_{\text{FR}}$ ⌊without reason⌋) in Sheridan's play *The Rivals* was famous for making such speech errors, as in

[7] Ironically, if we consider "mis-" as a negation prefix, the sentence can be interpreted as "they overestimated me."

a nice arrangement of *epitaphs

where the word she meant is "epithets." An experience we all have from time to time, namely the *tip-of-the-tongue* (TOT) experience, sheds additional light on the structure of the mental lexicon. Already in 1890, James [24, p. 251] wrote:

> the rhythm of the lost word may be there without the sound to clothe it; or the evanescent sense of something which is the initial vowel or consonant may mock us fitfully, without growing more distinct.

Brown [11] enumerates the TOT's characteristics: (a) it is a nearly universal experience, (b) it occurs about once a week, (c) it increases with age, (d) it is frequently elicited by proper names, (e) it often enables access to the target word's first letter, (f) it is often accompanied by words related to the target, and (g) it is resolved during the experience about half of the time. A pathology of being in a permanent TOT state is called *anomic aphasia*.

As for *grammatical words*, they are stored in a separate part of the mental lexicon and become available later when the grammatical sentence frame is constructed.

Building Words

After a lemma has been retrieved from the mental lexicon, it sometimes must be inflected to fit in a sentence. Inflection and derivation can be quite complex, so one may ask whether derived forms are always created on the fly or whether they become individual entries of the mental lexicon. Exchange errors such as

I just *sounded to *start

(instead of "I just started to sound") seem to imply that, at least for simple cases, inflection occurs *after* lemma selection. The fact that children may utter forms like

Yesterday I *swimmed in the pool

is not a counterexample. It merely shows that during their language development, they internalize morphological rules for conjugation and apply them indistinctly to all verbs until they learn the various exceptions.

In 1958, in a well-known experiment by the American psychologist Berko [6], known as the *Wug test*, various questions leading to the derivation of non-existing words were asked to preschool and first-grade children. Here are some of these questions:

1. Plural. [One bird-like animal, then two.] "This is a wug /wʌg/. Now, there is another one. There are two of them. There are two ___."

8. Derived adjective. [Dog covered with irregular green spots.] "This is a dog with quirks /kwərks/ on him. He is all covered with quirks. What kind of dog is he? He is a ___ dog."

14. Past tense. [Man dangling an object on a string.] "This is a man who knows how to bod /bad/. He is bodding. He did the same thing yesterday. What did he do yesterday? Yesterday he ___."

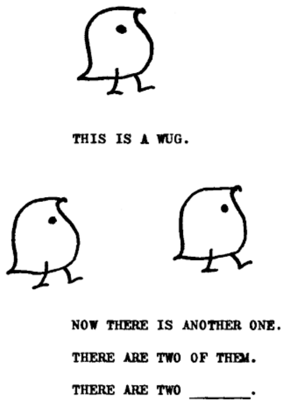


Fig. 4.1 An image from the Wug experiment by Berko [6]

> 19. Third person singular. [Man holding an object.] "This is a man who knows how to loodge /luwdž/. He is loodging. He does it every day. Every day he ___."

The results demonstrated that

> there can be no doubt that children in this age range operate with clearly delimited morphological rules. [6, p. 171]

The results were impressive for the wug question, a bit less for other cases:

Expected form	Pre-school	First-graders
1. wugs	76%	97%
8. quirky	0%	0%
14. bodded	14%	31%
19. loodges	57%	56%

where the first column is the percentage of correct answers by preschool children and the second is the one of first-graders. In the case of "quirky," 64% of the children formed the compound "quirk dog," but none gave the expected "quirky" answer. In the case of "bodded," most children answered "bod."

An argument in favor of on-the-fly form building is the existence of *morpheme shift* errors, as the following:

> I point *outed that
>
> He go *backs to

In these cases, the past-tense and 3rd person suffixes are applied to multiword units "point out" and "go back," which implies that these are stored as verb entries and then inflected as such.

Derivational morphology seems to use the same mechanisms as inflectional morphology (after all, the distinction between the two is not very crisp; see Tuggy [42]). In the following example:

Fancy getting your model *renosed

instead of "your nose remodeled," the four morphemes ABCD (A = nose, B = re, C = model, D = ed) have been reordered as CBAD, i.e., the stems have been exchanged while the suffixes have been kept. This calls for assembling the morphemes *after* lexical selection (and after the exchange of lemmas has occurred), similarly to the case of inflected derivation.

Phonological Encoding

After lemmas are filled into sentence slots and function words and grammatical morphemes are filled in, the next step is the generation of a *phonetic plan* that drives the speech organs.

The existence of this additional step is, once again, revealed by speech errors. In this case, the errors can be *substitutions* of phonemes, as in "*inimicable style"; *misordering* of phonemes: *anticipation*, where a phoneme comes earlier than expected, as in "*leading list" instead of "reading list," *perseverance*, where a phoneme reappears in the following word, as in "black bloxes," or *exchange*, where phonemes are exchanged, as in "knife light" instead of "night life"; *omission* and *addition*.

These phenomena occur *after* morphological processing. It has been observed that misorderings respect syllable structure: the affected syllables will either be both stressed or both unstressed (as in "*sedden *duth" for "sudden death," where the exchanged onsets belong to stressed syllables and the unstressed syllable is unaffected). Reverend William Spooner (1844–1930) was famous for exchanging initial phonemes, like in

the Lord is a *shoving *leopard

(instead of "the Lord is a loving sheppard"), therefore they are nowadays called *spoonerisms*.

Other features of phonological encoding errors are

(a) the fact that they are more frequent between phonetically close phonemes (as in "*par *cark," in which both /k/ and /p/ are voiceless stops) and
(b) errors that result in existing words are more frequent than those that result in nonwords.

The first feature may be due to an error in phonetic planning (since the "instructions" sent to speech organs will be pretty similar and hence more error-prone), and the second feature may hint towards a last-moment re-verification in the mental lexicon, blocking at least some nonwords and hence favoring existing words. For example, the spoonerism above results in a nonword ("*shoving") and an existing word ("leopard").

Hunting speech errors may be a thrill, but one will never find sufficient errors to establish decent statistics. To remedy the scarcity of spontaneous speech errors, experimental approaches use *tongue twisters* carefully built to induce a given type of error. For example, Wilshire [44] has used a corpus of 12,288 utterances of tongue twisters (such as “case port bed moon” for control and “palm neck name pack” and “moss knife noose muff” as alliterations) and obtained an average of 4.53% of erroneous utterances out of which he was able to perform statistics.

Keylogs

We saw how the dynamic speech process, with its errors, toils, and snares, can provide insight into how linguistic data is structured in the mind. Obviously, this cannot be done with written text, which is static. There is one case, though, where the dynamics of written text production can nevertheless be studied: you just need to connect a computer keyboard to a chronometer, ask the subject to keyboard a text, and store the precise timestamp of every keystroke.

Studies in German [43] and English [5] have shown that pauses between keystrokes are proportional to the level of linguistic structure boundaries: the shortest pauses are between graphemes, and they become increasingly longer between syllables, words, and constituents. In German, there was a clear difference between component boundaries, syllable boundaries, and graphemes inside syllables. For example, Weingarten [43, p. 539] reports that a semi-skilled typist (58 words/m) needs on average 110 ms longer to type the graphemes <ls> when they are located between components (as in “Rollschuh”$_{\text{DE}}$, or “Schaukelstuhl”$_{\text{DE}}$) than when they are located inside a component (as in “Halstuch”$_{\text{DE}}$, “falsch”$_{\text{DE}}$).

One can conclude that linguistic structures such as syllables, components, words, and constituents are indeed involved in how linguistic data is stored in our minds and processed by them.

4.11 Further Reading

4.11.1 Literature

Booij’s *The Grammar of Words* [9], in the *Oxford Textbooks in Linguistics* series, is a very clear and pleasant-to-read account of morphology. Another very thorough and well-organized textbook on Morphology is Lieber’s *Introducing Morphology* from *Cambridge Introductions to Language and Linguistics* [26]. As the 3rd edition of this book is very recent (2022), it offers an actualized version of the Wug test:

> Speakers of English use the suffixes -ize (crystallize) and -ify (codify) to form verbs from nouns. If you had to form a verb that means ‘do something the way Donald Trump does

> it,' which suffix would you use? How about a verb meaning 'do something the way Joseph Biden does it'? »

As expected, the answers are "Trumpify," and "Bidenize."

For those interested in comparing the morphologies of various languages of the world seeking universal features, we recommend Bobaljik's *Universals in Comparative Morphology* [8]. For a deeper insight into the Semitic root and patterns system, see Rubio [35]. The easiest way to discover Lojban is by reading Nicholas & Cowan [31].

4.11.2 Science Fiction

Orwell's Newspeak

One of the best illustrations of the use of morphology in Science Fiction can be found in the Appendix to Orwell's *Nineteen Eighty-Four*, called "The Principles of Newspeak". Newspeak has some fascinating properties: first of all, irregular inflected forms are prohibited: "brought" becomes "bringed," "spoke" becomes "speaked," "men" becomes "mans," etc. Derivation is also prohibited. One form is chosen (a verb or a noun) and used in all parts of speech. For example, "thought" is prohibited and "think" is used instead, as noun. There is only one suffix for building adjectives and one suffix for building adverbs, namely "-ful" and "-wise," so that we get "thinkful" and "thinkwise". There is also a single negation prefix, "un-," and a single emphasis prefix "plus-," so that weather could be "uncold," "cold," "pluscold," or "pluspluscold." Compound words were created from authorized components, for example "bellyfeel" is a feeling that implies a "blind, enthusiastic acceptance". Orwell quotes a leading article's headline: OLDTHINKERS UNBELLYFEEL INGSOC, where "unbellyfeel" is the opposite of the blind, enthusiastic acceptance of "bellyfeel" (the reader can imagine how brutal the consequences of an "unbellyfeel" of the regime can be…) and "oldthinkers" are those that dare to think in the old way, enemies of the regime. As for "Ingsoc," it is Big Brother's regime, a contraction of "English Socialism". In Newspeak, death is called "unlife".

Time Travel and Verb Morphology

Our next example contrasts with the brutal dystopia of Ingsoc. It is an excerpt from Douglas Adams's *The Restaurant at the End of the Universe*, the second volume of the trilogy of five books:

> The Restaurant at the End of the Universe is one of the most extraordinary ventures in the entire history of catering. It has been built on the fragmented remains of ... it will be built on the fragmented ... that is to say, it will have been built by this time, and indeed has been—
>
> One of the major problems encountered in time travel is not that of accidentally becoming your own father or mother. […] The major problem is quite simply one of grammar, and the

> main work to consult in this matter is Dr Dan Streetmentioner's *Time Traveller's Handbook of 1001 Tense Formations*. It will tell you, for instance, how to describe something that was about to happen to you in the past before you avoided it by time-jumping forward two days in order to avoid it. The event will be described differently according to whether you are talking about it from the standpoint of your own natural time, from a time in the further future, or a time in the further past and is further complicated by the possibility of conducting conversations whilst you are actually traveling from one time to another with the intention of becoming your own father or mother.
>
> Most readers get as far as the Future Semi-Conditionally Modified Subinverted Plagal Past Subjunctive Intentional before giving up, and in fact, in later editions of the book, all the pages beyond this point have been left blank to save on printing costs.

Time travel has not been invented yet, so we don't need the 1,001 tense formations for the moment. Adams translated the logical contradiction of time travel to the field of morphology. Thanks to his unique sense of humor, it seems perfectly natural. He also re-invented the book made of blank pages, which was initially invented by Cocteau, in his movie *Orphée* [20, pp. 333–334], as also mentioned in Dworkin's excellent analysis of emptiness in art [13].

The Golem

What we also encounter in Science Fiction is Semitic morphology, more precisely in the history of the Golem. The story unfolds as follows: a rabbi builds a human-like form from clay or mud and writes the word "אמת"$_{\mathrm{HE}}$ ⌊truth⌋, on his front (in the *X-Files* episode *Kaddish* S4E15, the word is written on the golem's hand). This creature, called Golem, gets alive and performs acts of revenge. It is undestroyable. To make it return to dust, the rabbi wipes out the first letter "א"$_{\mathrm{HE}}$, and what remains is "מת"$_{\mathrm{HE}}$ ⌊dead person⌋. The Golem returns to dust.

Let us investigate the morphology of these two words. "מֵת"$_{\mathrm{HE}}$ appears 78 times in the Bible. It is the active participle masculine singular absolute form of the root "מ-ו-ת"$_{\mathrm{HE}}$, i.e., we have a phenomenon of root contraction (the R_2 has vanished). As for "אֱמֶת"$_{\mathrm{HE}}$ ⌊truth⌋, it appears 127 times in the Bible and is, in fact, a contraction of "אֱמֶנֶת"$_{\mathrm{HE}}$ (notice the "נ"$_{\mathrm{HE}}$ letter at 3rd position), the root of which is "א-מ-נ"$_{\mathrm{HE}}$, and the most paradigmatic word of this root is "אָמֵנ"$_{\mathrm{HE}}$ ⌊truly⌋. The latter is nothing more than the adverb "Amen" used by all three book religions. The removal of the aleph letter from "אֱמֶת"$_{\mathrm{HE}}$ ⌊truth⌋ indeed gives "מֵת"$_{\mathrm{HE}}$ ⌊dead person⌋, and this works whether we write the words in voweled or unvoweled form. Furthermore, the removed letter aleph, in Jewish mysticism, represents the oneness of God; the symbolism is strong, and this has contributed to the popularity of the Golem legend.

4.12 Exercises

Exercise 3-1: English and French Verb Conjugation Compared

Find the English and French verbs that have the most complex conjugation. Find the English and French verb forms most distant from the infinitive form.

Exercise 3-2: Jules Verne and French Verbs

Fetch the earliest book and the latest book of Jules Verne that are available on Project Gutenberg[8] in the French language, as well as equally distributed intermediate books. Select the tools with the best performance on French morphology among *Talismane*,[9] *stanza*, *TreeTagger*,[10] and *spaCy*. Parse the file and study the distribution of verb tenses in these texts. Has it changed over the years? Did the verbs he used become more straightforward or more sophisticated over the years?

Exercise 3-3: The Combinatorics of Neoclassical Morphemes

We want to ask whether we can extract neoclassical prefixes and suffixes combina-
☞ 73 torially from a list of neoclassical compounds words? We will evaluate our results vs. an inventory of 990 Greek prefixes and suffixes and 1,035 Latin prefixes and suffixes.

Exercise 3-4: The Morphology of Lojban

Describe the morphology of the words of the following text in the Lojban language:

> ni'o seldau lo selmi'ecatra fa lo du'u na ka'e katna vimcu lo stedu se cau lo nu da poi xadni zo'u ka'e katna vimcu lo stedu da

which is the translation[11] by .xorxes. (Jorge Llambías) of a sentence of Lewis Carroll's *Alice's Adventures in Wonderland*:

> The executioner's argument was that you couldn't cut off a head unless there were a body to cut it off from.

adapted to our needs by Martin Bays.

---

[8] https://www.gutenberg.org/

[9] https://github.com/joliciel-informatique/talismane

[10] https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[11] https://alis.lojban.org

Exercise 3-5: Long and Round Ess in German

> Halte feſt an deutſchem Sinn, deutſchen Buchſtaben, denn wenn das Ding ſo fort geht, ſo wird in fünfzig Jahren kein Deutſch mehr geredet oder geſchrieben, und Du und Schiller, Ihr ſeid hernach klaſſiſche Schriftſteller wie Horaz, Livius, Ovid und wie ſie alle heißen, denn wo keine Sprache mehr iſt, da iſt auch kein Volk.[12]
>
> Katharina Eliſabeth Goethe

As the reader can see in the following excerpt from Goethe's *Dichtung und Wahrheit*,

> Solche Korreſpondenzen, beſonders mit bedeutenden Perſonen, wurden ſorgfältig geſammelt und alsdann, bei freundſchaftlichen Zuſammenkünften, auszugsweiſe vorgeleſen; und ſo ward man, da politiſche Diskurſe wenig Intereſſe hatten, mit der Breite der moraliſchen Welt ziemlich bekannt.

when German is written in broken script (in our case, Fraktur [19, 22, 38, 4]), the letter ess takes two forms: the "long ess" "ſ" (U+017F LATIN SMALL LETTER LONG S) and the "round ess" "s" (encoded as the usual ASCII "s"). The latter is used at word begin or word interior, the former is used at word end. The difficulty arises from the fact that by "word" we mean also components/morphemes, and therefore a round s can also appear in the word's interior, when it happens to be the last letter of a component (and this is frequently the case when the component is in genitive case). Here is an example:

"Reichshofratsagent"$_{\text{DE}}$ ⌊Imperial Council Agent⌋ → Reichshofratsagent

Build an engine for detecting long/round ess letters and apply it to Goethe's *Dichtung und Wahrheit* `dichtung.txt`. Use the golden corpus `dichtung-gold.txt` to evaluate the performance of your classification algorithm.

References

[1] Jean Aitchison. *Words in the Mind: An Introduction to the Mental Lexicon*. 4th ed. John Wiley & Sons, 2012.

[2] Pierre J.L. Arnaud. "Target–error resemblance in French word substitution speech errors and the mental lexicon". In: *Applied Psycholinguistics* 20 (1999), pp. 269–287.

[3] R. Harald Baayen. "Storage and computation in the mental lexicon". In: *The Mental Lexicon. Core Perspectives*. Ed. by Gonia Jarema and Gary Libben. Brill, 2007, pp. 81–104.

[12] "Hold tight to German sense, German letters, because if the thing goes on like this, in fifty years no more German will be spoken or written, and you and Schiller, you will be classical writers like Horace, Livius, Ovid and all their names, because where there is no more language, there is also no people." (Letter by Goethe's mother Katharina Elisabeth Goethe to her son, on December 25, 1807.)

[4] Peter Bain and Paul Shaw, eds. *Blackletter. Type and National Identity*. Princeton Architectural Press, 1998.

[5] Nicolas Ballier, Erin Pacquetet, and Taylor Arnold. “Investigating Keylogs as Time-Stamped Graphemics”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 353–365.

[6] Jean Berko. “The Child’s Learning of English Morphology”. In: *WORD* 14 (1958), pp. 150–177.

[7] Leonard Bloomfield. *Language*. London: Allen and Unwin, 1933.

[8] Jonathan David Bobalik. *Universals in Comparative Morphology. Suppletion, Superlatives, and the Structure of Words*. Current Studies in Linguistics. MIT Press, 2012.

[9] Geert Booij. *The Grammar of Words. An Introduction to Linguistic Morphology*. Oxford Textbooks in Linguistics. Oxford University Press, 2005.

[10] Dmitri Borgmann. *Beyond Language: Adventures in Word and Thought*. Charles Scribner’s Sons, 1967.

[11] James Cooke Brown. “Loglan”. In: *Scientific American* 202 (1960), pp. 53–63.

[12] John Woldemar Cowan. *The Complete Lojban Language*. A Logical Language Group Publication, 2016.

[13] Craig Dworkin. *No Medium*. Boston, MA: MIT Press, 2013.

[14] Umberto Eco. *The Search for the Perfect Language*. Translated by James Fentress. Fontana Press, 1997.

[15] Lutz Edzard. “Notes on the Emergence of New Semitic Roots in the Light of Compounding”. In: *Journal of Arabic and Islamic Studies* 17 (2017), pp. 407–414.

[16] Katerina T. Frantzi, Sophia Ananiadou, and Junichi Tsujii. “The C-value/NC-value Method of Automatic Recognition for Multi-word Terms”. In: *Proceedings of ECDL’98*. Vol. 1513. Springer LNCS. 1999, pp. 585–604.

[17] Lewis Glinert. *The grammar of Modern Hebrew*. Cambridge: Cambridge University Press, 1989.

[18] Cliff Goddard and Anna Wierzbicka. “Introducing Lexical Primitives”. In: *Semantic and Lexical Universals*. Ed. by Cliff Goddard and Anna Wierzbicka. Amsterdam/Philadelphia: John Benjamins Publishing Company, 1994.

[19] Yannis Haralambous. “Drucksatz in gebrochenen Schriften”. TEX-Tagung DANTE 2000, Technische Universität Clausthal, Mar 2000, Clausthal, Germany https://hal.archives-ouvertes.fr/hal-02111827/document. 2000.

[20] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. “Graphemic and Graphetic Methods in Speculative Fiction”. In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[21] Trevor A. Harley. “Speech errors: Psycholinguistic approach”. In: *The Encyclopaedia of Language and Linguistics*. Ed. by K. Brown. Vol. 11. Oxford: Elsevier, 2006, pp. 739–744.

[22] Wolfgang Hendlmeier. *Handbuch der Frakturschriften*. Hermann Schmidt Mainz, 1998.

[23] A. A. Isakova and K. V. Filina. “Tmesis in the context of expressive occasional word formation in the English language”. In: *Язык и мировая культура: взгляд молодых исследователей. Сборник материалов XV Всероссийской научно-практической конференции*. Novosibirsk State University, 2016, pp. 93–96.

[24] William James. *Principles of psychology*. New York: Holt, 1890.

[25] Frédéric Landragin. *Comment parler à un alien*. Le Bélial, 2018.

[26] Rochelle Lieber. *Introducing Morphology*. 3rd ed. Cambridge Introductions to Language and Linguistics. Cambridge University Press, 2022.

[27] Dilin Liu. In: *TESOL Quarterly* 44 (2010), pp. 4–30.

[28] Marie-Catherine de Marneffe et al. “Universal Dependencies”. In: *Computational Linguistics* 47 (2021), pp. 255–308.

[29] Andrew Nevins and Bert Vaux. "Metalinguistic, shmetalinguistic: the phonology of shm-reduplication". In: *Proceedings from the Annual Meeting of the Chicago Linguistic Society*. 2003, pp. 702–721.
[30] Nick Nicholas. Personal communication. 2022.
[31] Nick Nicholas and John Cowan, eds. *What is Lojban? .i la lojban. mo*. Logical Language Group, 2003.
[32] Jerome L. Packard. *The Morphology of Chinese. A Linguistic and Cognitive Approach*. Cambridge University Press, 2000.
[33] Muhamma Rafi'ia, I Wayan Pastika, and Ni Luh Nyoman Seri Malini. "Types of Indonesian Reduplication as the Translation Equivalence of English Lexicons". In: *Linguistika* 48 (2018), pp. 15–25.
[34] Armand S. Rotaru, Gabriella Vigliocco, and Stefan L. Frank. "From words to behaviour via semantic networks". In: *Proceedings of the 38th Annual Conference of the Cognitive Science Society (Cogsci 2016)*. Ed. by A. Papafragou, D. Grodner, and D. Mirman. 2016, pp. 2207–2212.
[35] Gonzalo Rubio. "Chasing the Semitic root: The skeleton in the closet". In: *Aula Orientalis* 23 (2005), pp. 45–63.
[36] Josef Ruppenhofer, Michael Wiegand, and Rebecca Wilm. "Distinguishing affixoid formations from compounds". In: *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, pp. 3853–3865.
[37] Karin C. Ryding. *A Reference Grammar of Modern Standard Arabic*. Cambridge: Cambridge University Press, 2005.
[38] Judith Schalansky. *Fraktur mon Amour*. Princeton Architectural Press, 2008.
[39] Frank A. Smadja and Kathleen R. McKeown. "Using collocations for language generation". In: *Computational Intelligence* 7 (1991), pp. 229–239.
[40] Michael Spagnol. "A Tale of Two Morphologies. Verb structure and argument alternations in Maltese". PhD thesis. Universität Konstanz, 2011.
[41] "The Miseducation of America". Time Magazine, Sunday, Jan. 11, 2009 https://content.time.com/time/specials/packages/article/0,28804,1870938_1870943_1870945,00.html.
[42] David H. Tuggy. "The inflectional/derivational distinction". Work Papers of the Summer Institute of Linguistics, University of North Dakota Session, Vol. 29. 1985.
[43] Rüdiger Weingarten, Guido Nottbusch, and Udo Will. "Morphemes, Syllables, and Graphemes in Written Word Production". In: *Multidisciplinary Approaches to Language Production*. Ed. by Thomas Pechmann and Christopher Habel. Vol. 157. Trends in linguistics studies and monographs. de Gruyter, 2004, pp. 529–572.
[44] Carolyn E. Wilshire. "The "Tongue Twister" Paradigm as a Technique for Studying Phonological Encoding". In: *Language and Speech* 42 (1999).

Chapter 5
Syntax

All grammars leak.

Edward Sapir, *Language: An Introduction to the Study of Speech*

Syntax is the subfield of linguistics that studies the grammaticality of sentences. But what is a sentence in the first place?

The language-related term "sentence" is among the most familiar terms to the layman as it is already introduced in first grade. Interestingly, it is quite problematic in linguistics. Even highly legitimate syntax textbooks, such as Sag [56] or Carnie [7], use the term "sentence" without ever defining it. Crystal's Dictionary [14] affirms that "Innumerable definitions of sentence exist" but gives only one: the "expression of a complete thought," which is rather vague and relies on the notion of "thought," which is not linguistic and therefore remains undefined in [14].

Others define a sentence in a grapholinguistic way, as a group of words starting with a capital letter (this works for bicameral scripts only) and ending at a *strong punctuation mark* (i.e., a period, exclamation mark, or question mark). This definition is quite convenient for detecting sentences in (well-written) text. Still, it gives us no information on the nature of a sentence and no criteria about where to place these punctuation marks during text production.

In the following, we will use building blocks called *clauses* and *phrases* to build sentences. Once a simple sentence is obtained (for example, a simple clause consisting of a verbal phrase and a noun phrase, as in "John runs."), it can be extended by additional building blocks to obtain arbitrarily long sentences. Probably the longest sentence in French literature can be found in Proust's *Swann's Way*:

> But instead of simplicity, it was to ostentation that I must assign the first place if, after I had compelled Françoise, who could hold out no longer and complained that her legs were 'giving' beneath her, to stroll up and down with me for another hour, I saw at length, emerging from the Porte Dauphine, figuring for me a royal dignity, the passage of a sovereign, an impression such as no real Queen has ever since been able to give me because my notion of their power has been less vague and more founded upon experience—borne along by the flight of a pair of fiery horses, slender and shapely as one sees them in the drawings of Constantin Guys, carrying on its box an enormous coachman, furred like a cossack, and by

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_5

> his side, a diminutive groom like Toby, "the late Beaudenord's tiger," I saw—or rather I felt its outlines engraved upon my heart by a clean and killing stab—a matchless victoria, built rather high and hinting through the extreme modernity of its appointments, in the forms of an earlier day, deep down in which lay negligently back M$^{\text{me}}$ Swann, her hair, now quite pale with one grey lock, girt with a narrow band of flowers, usually violets, from which floated down long veils, a lilac parasol in her hand, on her lips an ambiguous smile in which I read only the benign condescension of Majesty, though it was pre-eminently the enticing smile of the courtesan, which she graciously bestowed upon the men who bowed to her.
> (Translation by C. K. Scott Moncrieff.)

The effect of this 260-word long sentence is devastating. When exposed to it, we run out of breath, embodying the narrator's breathlessness in front of M$^{\text{me}}$ Swann's overwhelming beauty.

Turning to the other extreme, the shortest English sentence is reported to be:

> I am.

These two sentences have in common that they are made of *building blocks*, which we will consider in this section. We will consider two types of building blocks: *constituents* and *clauses*.

5.1 Constituents and Clauses

Take the sentence, "The Doctor saved Clara from the Daleks." There are five *phrases* (also called *constituents*) in this sentence (we mark them by indices):

> $[_1$The Doctor$]_1$ $[_2$saved $[_3$Clara$]_3$ $[_4$from $[_5$the Daleks$]_5]_4]_2$.

Phrases 1 and 5 are called *noun phrases* since they contain a determiner and a noun; phrase 2 is a *verb phrase* since it starts with a verb; phrase 3 is a *prepositional phrase* since it starts with a preposition.

5.1.1 Constituency Tests

The constituents we detected in the sentence above are apparent, but what happens if we are in doubt? Fortunately, there are tests for detecting constituents, and according to Osborne [51], constituency tests are indeed the essential tools with which syntacticians practice their trade. In this section, we will describe nine different tests. A sequence of consecutive words $w_1, \ldots, w_n$ may not pass all tests, but if it passes at least two tests, then we can be confident about it being a constituent. Here are the eight tests we have chosen as being the most important ones [7, pp. 98–99]:

1. the *topicalization test*: if words $w_1, \ldots, w_n$, can be moved to the beginning of the sentence to be emphasized, then $w_1, \ldots, w_n$ form a constituent.
   E.g., "The Doctor fought in Galifrey" becomes "In Galifrey the Doctor fought," so we can infer that "in Galifrey" is a constituent.

2. the *clefting test*: it is a variant of the previous test. If out of $w_1, \ldots, w_n$ we can build a pattern of the form "It is/was $w_1, \ldots, w_i$ that $w_{i+1}, \ldots, w_n$," then $w_1, \ldots, w_n$ is a constituent.
   Examples: "It was the Doctor that saved Clara from the Daleks," "It was the Daleks that the Doctor saved Clara from," etc.;
3. the *replacement test*: if we can replace $w_1, \ldots, w_n$ by a single word w' of part-of-speech A, then $w_1, \ldots, w_n$ is a constituent, and more precisely an A-phrase. In "The Doctor saved Clara from the Daleks," we can replace "saved … Daleks" with "smiled" to obtain "The Doctor smiled." Therefore, "saved Clara from the Daleks" is a constituent, and since "smiled" is a verb, this constituent is a verb phrase;
4. the *pronoun* or *do-so substitution*: this is a variant of the previous one: if we can replace $w_1, \ldots, w_n$ by a pronoun or by some form of the expression "to do so," then $w_1, \ldots, w_n$ is a constituent.
   Using the same example, by writing "He saved her from them," we infer that "the Daleks" is a constituent; we can also write "The Doctor did so," where "did so" stands for "saved Clara from the Daleks," so that the latter is a constituent;
5. the *answer fragment*: if there is a question of which $w_1, \ldots, w_n$ is the answer, then $w_1, \ldots, w_n$ is a constituent.
   To the question "From whom did the Doctor save Clara?" we get the answer "from the Daleks," so the latter is a constituent;
6. the *stand-alone test*: it is a variant of the previous one. If there is a question of which $w_1, \ldots, w_n$ (or $p, w_1, \ldots, w_n$, with p a pronoun) is a stand-alone answer, then $w_1, \ldots, w_n$ is a constituent.
   The question "What did the Doctor do?" can be answered by "(he) saved Clara from the Daleks," so that "saved … Daleks" is a constituent. Also "Who saved Clara?" can be answered by "The Doctor," and "Whom did the Doctor save?" by "Clara," so that "The Doctor" and "Clara" are constituents;
7. the *passive test*: if from a sentence "$w_1, \ldots, w_i\ V\ w_{i+1}, \ldots, w_j\ w_{j+1}, \ldots, w_n$," where V is a verb in active form, we can build a new sentence "$w_{i+1}, \ldots, w_j\ V'\ P\ w_1, \ldots, w_i\ w_{j+1}, \ldots, w_n$," where V' is in passive form, and P a preposition, then $w_1, \ldots, w_i$, $w_{i+1}, \ldots, w_j$ and $w_{j+1}, \ldots, w_n$ are constituents.
   In our case, the passive form "Clara was saved by the Doctor from the Daleks" exhibits three constituents: "Clara," "the Doctor" and "from the Daleks";
8. the *coordination test*: if in a sentence, after $w_1, \ldots, w_n$ we can add a conjunction "and" followed by an A-phrase, then $w_1, \ldots, w_n$ is a constituent and an A-phrase. This is probably the most popular constituency test.
   Examples: "[The Doctor] and [K9] saved Clara from the Daleks," where "K9" is a noun, therefore "The Doctor" is a noun phrase; "The Doctor saved Clara from [the Daleks] and [the Cybermen]," therefore "the Daleks" is a noun phrase, etc.

None of the above tests will work with the word sequences "Doctor saved" or "Clara from" or "from the," therefore they cannot be constituents. Carnie [7, p. 100] gives an example of a word sequence that passes the conjunction test but no other test. As a caveat: in "The Doctor loves, and the Master hates humanity," one could say that

“The Doctor loves” and “The Master hates” pass the conjunction test—nevertheless, they pass no other test, so they cannot be constituents.

Besides noun phrases, verb phrases, and prepositional phrases, there are also *adjectival phrases* and *adverbial phrases*. In the sentence “The Doctor saved the lovely Clara bravely,” “lovely” will be an adjectival phrase and “bravely” an adverbial phrase. But what about “The Doctor saved the very lovely Clara”? Is “very lovely” an adjectival or an adverbial phrase? To settle this, we define the notion of *head* and *modifier*. In the sentence, “very” is a modifier of “lovely,” so the phrase “very lovely” takes the type of “lovely” and therefore is an *adjectival* phrase. A phrase always has the type of its head.

5.1.2 Agreement

> There is nothing more likely to start disagreement among people or countries than an agreement.
>
> E.B. White, *One Man’s Meat*

The actions of an individual in a society affect other individuals. This is also the case in a sentence: inflecting a lexeme often leads to the inflection of other lexemes. The typical example is the interaction between verb and pronoun: when the subject pronoun is in the third person, the verb following it must also be in the third person: “I eat” becomes “*She* eat*s*.” This phenomenon is called *agreement*.

As inflection in English is limited compared to other languages, agreement in English is, in some sense, a non-issue. The situation is entirely different in French, where the rules of agreement are so complex that even native speakers have a hard time deciding whether agreement has to be applied or not[1]. But there is a bright side of this phenomenon: agreement allows disambiguation of syntax trees, as in “la maison au balcon bleu”$_{\text{FR}}$ where the masculine adjective can only modify the word “balcon”$_{\text{FR}}$ and not the word “maison”$_{\text{FR}}$, which is feminine so that the correct interpretation is ⌊the house with the blue balcony⌋ and not ⌊the blue house with a balcony⌋.

In most Western languages, the verb agrees with its subject concerning number and person: “a woman speak*s*,” “the wom*e*n speak.” An exception is made for *collective nouns* as in “The committee have not yet decided.”

Interestingly, Attic (ancient) Greek collective nouns behave in precisely the opposite way: the noun is in the plural and the verb in the singular, as in “τὰ παιδία παίζει”$_{\text{EL}}$ ⌊the children play⌋ (instead of “παίζουσιν”$_{\text{EL}}$ which would be the plural form of the verb)—in fact, there is flexibility between the singular and the plural forms

[1] A typical example that hardly any speaker of French is aware of is the case of verbs such as “se succéder”$_{\text{FR}}$ ⌊to follow one another⌋, “se plaire”$_{\text{FR}}$ ⌊to like each other⌋, “se parler”$_{\text{FR}}$ ⌊to talk to each other⌋ for which there is no pronominal agreement since their complement is always indirect: in “Les familles se sont succédé”$_{\text{FR}}$ ⌊One family followed another⌋ (instead of “succédées”$_{\text{FR}}$) there is no agreement of the past participle because each family is the indirect compliment of the other. A study in psychology has shown that cognitive overload increases agreement errors in French [19].

of the verb, and sometimes the form that fits best with the metric structure is chosen [58, §959a].

In Arabic, besides number and person, the agreement between subject and verb also depends on *gender*. There is a different form of the verb for masculine and feminine subjects: “ينام احمد”$_{\text{AR}}$ ⌊Ahmed sleeps⌋ (Ahmed is a masculine name) vs. “تنام لونا”$_{\text{AR}}$ ⌊Luna sleeps⌋ (Luna is feminine).

From a cognitive point of view, agreement is redundant information that consolidates information processing (see the very interesting study [63] for a survey of the importance of redundancy in language). Depending on language, agreement may use more or less morphemes—take for example the phrase “the beautiful little dancer♀” (meaning a female dancer): in English and in Swedish (“en vacker liten dansare”$_{\text{SE}}$) there is no gender-specific morpheme, in Danish there is one morpheme (“en smuk lille danserinde”$_{\text{DK}}$), in German and Hebrew there are two morphemes (“die schöne kleine Tänzerin”$_{\text{DE}}$, “הרקדנית הקטנה היפה”$_{\text{HE}}$), in Russian and Arabic there are three morphemes (“красивая маленькая танцовщица”$_{\text{RU}}$, “الراقصة الجميلة الصغيرة”$_{\text{AR}}$), and in French and Greek there is maximum redundancy, through the use of four gender-specific morphemes (“la belle petite danseuse”$_{\text{FR}}$ “ἡ ὄμορφη μικρὴ χορεύτρια”$_{\text{EL}}$).

5.1.3 Clauses

A noun phrase followed by a verb phrase is called a *clause* or a *tense phrase*. Between the noun phrase and the verb phrase, there may be a modal verb (e.g., “may”) or an auxiliary (e.g., “has”). Their part-of-speech class is called *tense*. Clauses can be combined, either by conjunction or by embedding. In the latter case, we can use *complementizers* such as “that” or “if” to introduce embedded clauses. Here is an example of a sentence containing three embedded clauses:

Clara said *that* $_1$[she heard $_2$[the Doctor say *that* $_3$[he is from Galifrey]$_3$]$_2$]$_1$.

5.1.4 Topology

In syntax, by *topology* is meant the order of the three most important constituents of a clause, namely the subject (S), the verb (V), and the object (O). There are languages such as English and French where this order is invariably SVO (“the boy eats the apple,” “le garçon mange la pomme”$_{\text{FR}}$). In interrogative sentences it may become VSO (“is this true?,” “parlez-vous français?”$_{\text{FR}}$).

Yoda is a character from the *Star Wars* franchise. As can be verified in the Yoda corpus[2], the Yoda pseudo-syntax uses the word-order OSV (e.g., “my home this is,” “rootleaf I cook”), which occasionally becomes OVS in interrogative sentences (“afraid are you?”), even though not all interrogative sentences are OVS (e.g., “victory you say?” is OSV). There is a popular belief that Yoda syntax is inspired by

2 https://www.kaggle.com/datasets/stefanocoretta/yoda-speech-corpus

Japanese syntax. Unfortunately, we have to argue against it. Yoda syntax is OSV, while real-world Japanese is SOV. It suffices to compare the Yoda sentence "my home this is" with the Japanese "ここが私の家です"$_{\text{JA}}$ ⌊this my home is⌋. See [41] for a finer analysis of Yoda syntax.

Classical Arabic will place the verb first and therefore use a VSO order, but modern Arabic tends to use rather SVO order. Irish also uses VSO order: "Shábháil Clara an Dochtúir"$_{\text{IE}}$ ⌊saved Clara the Doctor⌋.

Languages with case marking tend to have more flexible word orders since S can be distinguished from O through morphological features.

And finally, as the most extreme case, the artificial language Lojban is defined so that *any* word order can be used! Nevertheless, SVO is the most commonly used order in Lojban, probably because of the influence of the native tongues of Lojban speakers on their use of Lojban.

5.1.5 Ambiguity

It turns out that sentence ambiguity is sometimes due to multiple syntax patterns that can be applied to them. The paradigmatic example is the sentence "I saw a man with a telescope," which can be interpreted as "I used a telescope to see a man" or as "I saw a man that was carrying a telescope." In the first case, the constituents are "[I] [saw [a man]] [with [a telescope]]," in the second case, they are "[I] [saw [a [man [with [a telescope]]]]]."

5.2 Syntax Theories

> The linguistic landscape is littered with literally hundreds if not thousands of theories of syntax, many with no more than a handful of adherents.
>
> Kuiper & Nokes, *Theories of Syntax*

The previous sections were an *avant-goût* of how syntax models a sentence's structure. Phrases and clauses are familiar to us because they are taught in high school. But they are only the tip of the iceberg. In the twentieth (and early twenty-first) century, there have been many attempts to describe a sentence's structure and give means to decide whether a given sentence is grammatical or not. Here is a short list of the most prominent theories in chronological order (the date and reference chosen are the ones of the first publications of the subject):

1935 Ajdukiewicz's Categorial Grammars [1];
1953 Bar Hillel's contribution to Categorial Grammars [2];
1954 Tesnière's Dependency Grammars [64, 65];
1957 *Chomsky's Context-Free Phrase Structure Grammars (CFG) [10], §5.3;

1958	Lambek's contribution to Categorial Grammars (DG) [43];
1960	*Hays's contribution to Dependency Grammars [30], §5.9;
1962	Martinet's Functional Syntax [49];
1965	*Chomsky's Transformational Grammar [8], §5.4;
1970	Woods's Augmented Transition Networks [68];
1974	*Chomsky and Jackendoff's $\overline{\mathrm{X}}$ Theory [35], §5.6;
1975	Joshi's Tree Adjunct Grammars [37];
1977	Pullum's Relational Grammars [54];
	Lakoff's contribution to Construction Grammars [42];
1979	Kay's Functional Unification Grammars [39];
1980	Dik's Functional Grammar [16];
	Van Valin's Role and Reference Grammars [66];
1981	*Chomsky's Government and Binding theory [9], §5.5;
	Kaplan & Bresnan's Lexical-Functional Grammar [38];
1985	Gazdar's Generalized Phrase Structure Grammars [22];
	Halliday's Systemic Functional Grammars [25];
1987	*Pollard & Sag's Head-Driven Phrase Structure Grammars (HPSG) [53], §5.7;
	*Steedman's Combinatory Categorial Grammars (CCG) [61], §5.8;
	Langacker's Cognitive Grammars [44];
1988	Hopper's Emergent Grammar [34];
	Fillmore's contribution to Construction Grammars [20];
1990	Legendre's Harmonic Grammars [46];
1995	Chomsky's Minimalist Program [11];
2001	Legendre's applications of Optimality Theory to syntax [45];
	Croft's Radical Construction Grammar [13];
2008	Hengeveld's Functional Discourse Grammar [32].

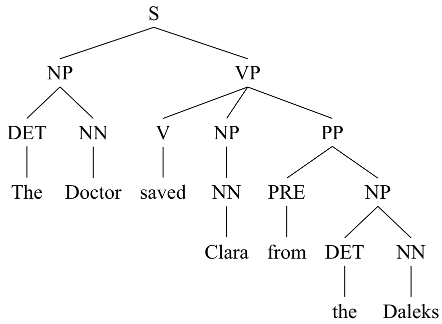
We have marked the theories we will briefly describe in this chapter with an asterisk. CFGs and DGs are extensively used in NLP, CCGs a bit less, but they are quite promising.

Before we start describing the various approaches to syntax, we would like to highlight an often neglected fact: the difficulty in defining syntax theories is not so much to be able to affirm that a sentence is grammatical but rather to be able to assert that a given sentence is *not* grammatical. For example, all theories that came before $\overline{\mathrm{X}}$ Theory were able to qualify "The book of poems with the blue cover" as a grammatical noun phrase, but $\overline{\mathrm{X}}$ Theory was the first to be able to assert that "The book with the blue cover of poems" is *not* a grammatical noun phrase (see §5.6).

5.3 Chomsky's Context-Free Phrase Structure Grammar

We have seen in Section 5.1 that constituents play an essential role in a sentence. What we need is a method to obtain them for any sentence. Our first attempt will be Chomsky's context-free grammar phrase structure (CFG).

Let us write the sentence “The Doctor saved Clara from the Daleks” as the leaves of a tree, the root of which we will call S (like “sentence”). Parent nodes of word nodes will be their *part of speeches*. And whenever a sequence of words forms a constituent, we will add a node and label it according to the constituent type. We thus obtain a syntax tree of the sentence, called the *constituent-tree*:



☞ 247 In his book *Syntactic Theories* [10], Chomsky describes a way to obtain formal grammar rules out of a syntax tree, as follows: S is the start axiom, the various POS tags are intermediate symbols, word nodes are terminal symbols, and if we denote by $Y_{i,1},\dots,Y_{i,n_i}$ the children of the node of symbol X_i, then we define the set of rewrite rules as $\{X_i \to Y_{i,1},\dots,Y_{i,n_i}\}$, for all i.

In the case of the example above, we obtain the following rewrite rules (where NN stands for a proper noun and ‘|’ is the “or” operator):

S → NP VP	DET → The | the
NP → DET NN | NN	NN → Doctor | Clara | Daleks
VP → V NP PP	V → saved
PP → PRE NP	PRE → from

The rules on the left are called *syntax rules* while the rules on the right are called *lexical rules*. Notice that the rule NP → DET NN is used twice in the tree.

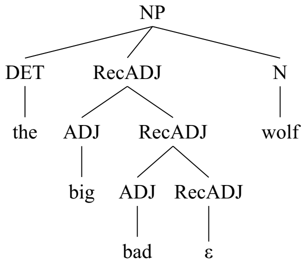
Chomsky’s fundamental hypothesis is as follows: there is, for each natural language, a context-free formal grammar with start symbol S, called the *phrase structure grammar of the language*, with the property that every sentence that derives from S in this formal language is *grammatical*, and every sentence that cannot be derived from it is *agrammatical*. ☞ 263

In other words, such a formal grammar can generate *all* grammatical sentences of a given language. The approach is therefore called a *generative* model of the language.

To limit the number of rewrite rules necessary to model a language, *recursion* is used. For example, to allow an arbitrary number of adjectives between a determiner and a noun, it suffices to introduce a special intermediate symbol we can call RecADJ and write the following rules (where ε is the empty word):

NP → DET RecADJ N
RecADJ → ε | ADJ RecADJ

The rule is recursive because RecADJ appears on both the left and right sides. If "big" and "bad" are ADJ and "wolf" is an N, then "the big bad wolf" is obtained through the following syntax tree (where ε is the empty word):

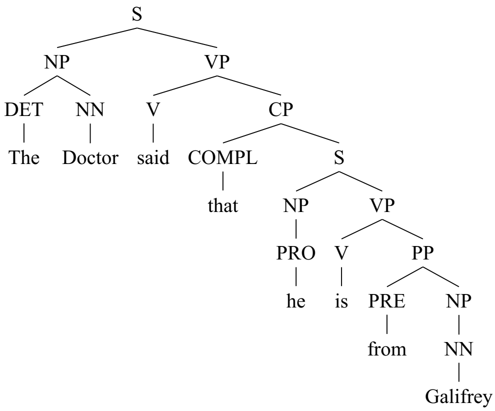


In the tree of the sentence "The Doctor saved Clara," we can detect three fundamental grammatical relations:

1. the *subject*: an NP located before the VP;
2. the *direct object*: an NP that is a child of the VP;
3. the *indirect object*: an NP that is child of a PP.

Of course, these rules are by no means exhaustive: in a sentence like "Amy gave Rory the keys," "Rory" is an indirect object even though there is no preposition (in a rephrasing of the same sentence, we get "Amy gave the keys to Rory," where there is indeed a preposition before "Rory").

To model sentences with subordinate clauses, we only need to add a POS type for the subordinate clause (CP) and a POS type for the potential completive (COMPL). Here is an example:



The number of verb arguments is called its *valency*. One of the limitations of CFGs is that they cannot handle phenomena such as verb valency and agreement unless, of course, we add new symbols for transitive, intransitive, and bitransitive verbs and for the very many features of nouns and verbs, which can lead to combinatorial explosion. For example, to ensure that "I sleep oranges" is agrammatical, while "I eat oranges" is grammatical, we need different POS labels for the verb "to sleep" (valency 0) and the verb "to eat" (valency 1); to ensure that the German "des Mannes"$_{\text{DE}}$ ⌊of the man⌋ (genitive singular) is grammatical while *"des Mann"$_{\text{DE}}$ is agrammatical, we need a POS tag for "des"$_{\text{DE}}$ that incorporates its case and number.

Not to mention cases where a verb can have multiple valency values such as "to ask": "I asked him a question" (valency 2), "I asked him" (valency 1), "All you need is to ask" (valency 0).

A reproach that could be made to CFGs is that they do not consider the lexicon. In later versions of his theory, Chomsky added support for *features*.

After a concise example of parsing context-free phrase structure grammar in Python, in the following sections, we will consider some improvements to generative grammar intended to solve specific grammaticality problems.

5.3.1 Parsing Context-Free Phrase Structure Grammar in Python

There are many constituency parsers out there. Among those implemented in Python, there is *stanza* [55]. Here is the code for parsing the two sentences of this section:

```
import stanza
stanza.download('en')
nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,constituency')
for text in ["The Doctor saved Clara from the Daleks",\
    "The Doctor said that he is from Galifrey"]:
    doc=nlp(text)
    for sent in doc.sentences:
        print(sent.constituency)
```

in which we first download the English module, then instantiate a pipeline requesting tokenization, POS tagging, and constituency, and finally parse the two sentences. The result is:

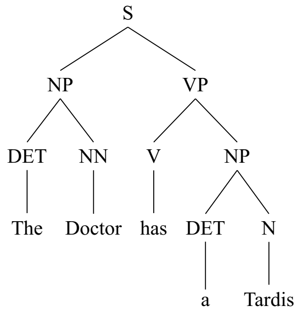
```
(ROOT (S (NP (DT The) (NNP Doctor)) (VP (VBD saved) (NP (NNP Clara))
(PP (IN from) (NP (DT the) (NNPS Daleks))))))

(ROOT (S (NP (DT The) (NNP Doctor)) (VP (VBD said) (SBAR (IN that)
(S (NP (PRP he)) (VP (VBZ is) (PP (IN from) (NP (NNP Galifrey)))))))))
```

In the second example, the node above the S node of the subordinate clause is written `SBAR`, otherwise $\overline{\text{S}}$, as we will see in § 5.6.

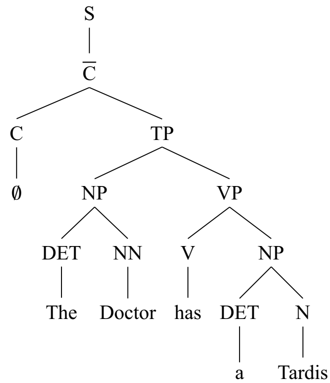
5.4 Chomsky's Transformational Grammar

Almost ten years after the release of *Syntactic Theories* [10], Chomsky published another groundbreaking book: *Aspects of the Theory of Syntax* [8]. One of its main ideas is that there are two levels of grammatical structures: *deep structure* (which is in direct connection with the mental lexicon) and *surface structure* (which is uttered).

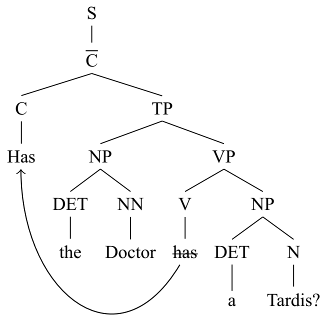
Furthermore, certain phenomena that would make CFG grammars too complex are handled not by individual trees but by *forests* obtained by tree transformations. The ☞ 210 typical example is *subject-auxiliary inversion* (SAI), which occurs when a declarative sentence is inverted into an interrogative one. Let us take the following sentence:



When we embed this sentence as a clause in a larger sentence, such as "Clara knows that the Doctor has a Tardis," a complementizer "that" is needed, it links the upper-level sentence and the embedded clause. Chomsky considers that there is an empty node at the location of the complementizer and that this empty node should carry a special part-of-speech tag C ("complementizer"):



(we will explain the TP and $\overline{\text{C}}$ tags in §5.6 and §5.6.1 respectively). This node is used when the clause is embedded *but*—and here comes the interesting part—the empty node can also be used as the target of a transformation between the interrogative and the declarative sentence. In other words, with a single syntax tree, we get two forms of the sentence (declarative and interrogative). It suffices to place the word "has" in one location and to leave the other location empty:



A tree becomes a "meta-tree," and a sentence, a "meta-sentence," containing several forms of itself.

In Chomsky's transformational grammar, tree transformations are also used for agreement. Instead of adding morphological features to POS tags, one can start with a tree of lemmas and then add suffixes and other grammatical morphemes through tree transformations.

5.5 Binding Theory

Chomsky has introduced binding theory [9] as well. Following Carnie [7], we will illustrate it by the solution of a specific problem, namely the one of *reflexive pronouns*, such as "himself," "themselves," etc. Let us ask the following question: What makes the sentence "the father of Heidi came himself to the party" grammatical while *"the father of Heidi came herself to the party" is agrammatical? Can this specific grammaticality property be deduced from the syntax tree?

The answer is yes, and we will see how in this section.

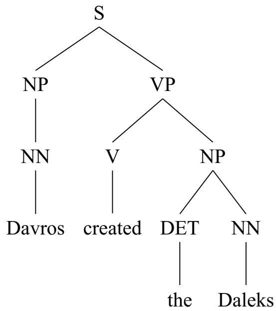
Let us start by defining structural relations between nodes [7].

5.5.1 Domination

In a syntax tree, we say that a node A *dominates* a node B if A is an ancestor of B. We say that a node A *exhaustively dominates* a sequence of terminal nodes $T_1, \ldots, T_n$,

- if A is an ancestor of all T_i,
- and if A is not an ancestor of any other terminal node.

In other words, if $T_1, \ldots, T_n$ is exactly the projection of A in the tree, using the direction that goes from the root of the syntax tree to the leaves. In the example

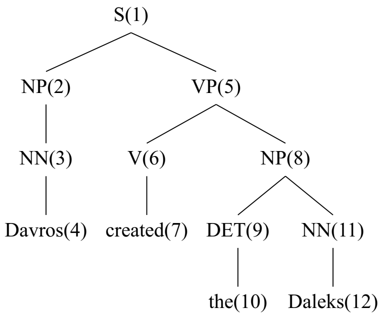


the nodes exhaustively dominated by the VP node are "created," "the," and "Daleks".

By construction of the syntax tree, a sequence of consecutive terminal nodes $T_1, \ldots, T_n$ is a *constituent* if and only if the nodes $T_1, \ldots, T_n$ are exhaustively dominated by the same node.

5.5.2 Precedence

Syntax trees are very special trees: not only are they *rooted* and *directed*[3] but they are also *ordered*: the natural order of words of the underlying sentence induces a total node order. The order is obtained by going downwards from the root, always taking the leftmost path and memorizing all locations where a choice has been made; once we arrive at the leaves, we go back to the closest memorized location, and we select the leftmost not selected child; we repeat this until all nodes have been traversed. The method is best understood by going through an example, so here is the same tree as in the previous section with numbers denoting the order of nodes:



Notice that, because you cannot cross branches in syntax trees,

1. the order of terminals as words in the sentence is compatible with the internal order of the tree,
2. S is always the first node, and

[3] We do not use the arrow notation that is common in the representation of digraphs, but we do draw the tree so that the downward direction denotes the ancestor-to-descendant relation.

3. the last terminal is always the last node of the tree.

Also, notice that whenever a node A *dominates* a node B, it also necessarily *precedes* it. Obviously, the reciprocal is false.

5.5.3 C-command

We say that a node A *c-commands* its siblings and the descendants of its siblings. The set of c-commanded nodes of A can be split in two parts: the first, called set of *symmetrically c-commanded* nodes, contains only siblings of A, and the second, called the set of *asymmetrically c-commanded* nodes, contains only descendants of siblings of A. In the "Davros created the Daleks" syntax tree of the previous section, V(6) symmetrically c-commands NP(8) and asymmetrically c-commands the four nodes DET(9), "the"(10), NN(11), and "Daleks"(12).

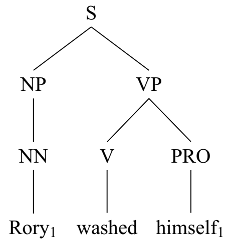
We will say that a node Y *intervenes* between nodes X and Z if X c-commands Y (either symmetrically or asymmetrically) and Y asymmetrically c-commands Z. For example, in the Davros tree, V(6) intervenes between NP(2) and DET(9) and DET(9) intervenes between V(6) and Daleks(12).

5.5.4 Referring Expressions, Anaphors, Binding

A *referring expression* is an NP that refers to some entity in the world (be it an abstract or a concrete entity, in the real world or some imaginary world). This is the case for most NPs we use in sentences. But there is a particular case of NP that necessarily refers to another NP in the sentence: it is the case of *reflexive pronouns* like "herself," "themselves," etc. In the general case of pronouns, the reference may be internal or external, e.g., in "She$_1$ said that she$_2$ is happy," the referent of "she$_2$" may be "she$_1$" or some other entity previously mentioned or designated by a gesture or by other means. But in the case of reflexive pronouns, no doubt is allowed: the reference *must* be internal. We call such a reference an *anaphora* and the target of the anaphora, its *antecedent*.

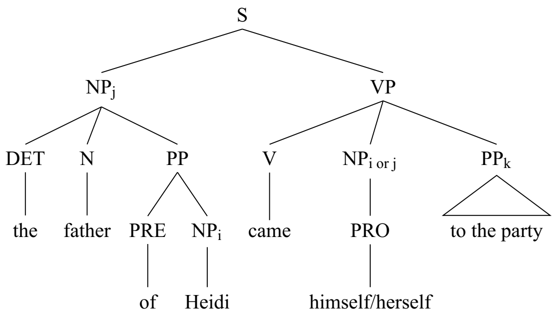
We will use the following notation: we will bracket NPs and add an index so that two NPs referring to the same entity will carry the same index—such NPs will be called *coindexed*, as in "[Heidi]$_i$ came [herself]$_i$ to [the party]$_j$."

Let A and B bet two NPs, and let A′ and B′ be nodes that exhaustively dominate A and B. If A′ c-commands B′ and if A and B are coindexed, then we will say that A′ *binds* B′. We will say that B′ is *bound* (without necessarily specifying by whom). Here is an example:



As can be seen in the tree, the NP node exhaustively dominates "Rory," and the VP node exhaustively dominates "himself"; also, NP c-commands VP since the latter is the sibling of the former, and finally "Rory" and "himself" are coindexed: in this case, NP binds VP, and consequently, VP is bound.

The notion of *binding* gives us a way to decide which of the two sentences "[the father of [Heidi]$_i$]$_j$ came [himself]$_j$ to [the party]$_k$" and "[the father of [Heidi]$_i$]$_j$ came [herself]$_j$ to [the party]$_k$" is grammatical. Here is the tree of these two sentences (the triangle stands for a part of the tree that we don't wish to develop):

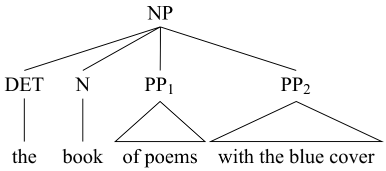


Binding theory [9] provides us with the rule that *anaphoras must be bound by their antecedent*. From the tree, it is clear that NP$_j$ (first child of S) indeed c-commands NP$_{i \text{ or } j}$, but NP$_i$ (parent of "Heidi") does not c-command NP$_{i \text{ or } j}$ and therefore cannot possibly bind it. Therefore, the only possible antecedent of NP$_{i \text{ or } j}$ is NP$_j$, and "himself" is the only possible choice since the real-world referent of NP$_j$ is masculine.

This is, of course, only a simplistic example. Still, our purpose was to show how introducing a new property (in this case: *coindexation*) and its interaction with the structural properties of the syntax tree allows us to detect agrammatical sentences in a much finer way than using the syntax tree alone.

5.6 $\overline{\text{X}}$ Theory

Let us take the NP "the book of poems with the blue cover," which is grammatical:

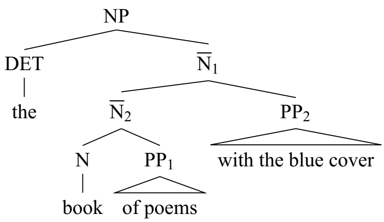


In this section, we will see how CFGs can be improved to detect the agrammaticality of the NP *“the book with the blue cover of poems,” where we have simply exchanged PP_1 and PP_2. This operation does not affect grammaticality if we use CFG theory only.

Let us apply the unitary replacement test to all subsequences of “the book of poems with the blue cover,” as in “I bought [the book of poems] with a blue cover, not the book of math with a red cover.” We obtain the following NPs: “the book of poems with a blue cover,” “book of poems with a blue cover,” “book of poems,” and “book.” But in the syntax tree, only the first (“the book of poems with a blue cover”) and the last (“book”) are marked as constituents.

Why not have a separate node in the tree for each of these constituents? We will add extra intermediate nodes and give them special tags: as all of them have a common head, namely the word “book,” which is an N, we will use the tag $\overline{\mathrm{N}}$. We use the new tag $\overline{\mathrm{N}}$ to signify that these nodes are neither N nor NP but “something in between.” In other cases, we would proceed similarly with other head types and obtain $\overline{\mathrm{V}}$, $\overline{\mathrm{ADJ}}$, etc. It is because of this graphemic notation that the theory is called $\overline{X}$ *Theory* (since a bar[4] can be placed on any tag X).

The $\overline{\mathrm{N}}$-extended tree becomes like this:



This trick allows to structurally distinguish PP_1 and PP_2: whenever we have a rewrite rule of the kind $\overline{\mathrm{N}} \to \overline{\mathrm{N}}$ PP, the PP will be called an *adjoint*, and whenever we have a rewrite rule $\overline{\mathrm{N}} \to$ N PP, the PP will be called a *complement*. $\overline{\mathrm{X}}$ theory states the rule that *complements must be closer to the noun than adjoints*.

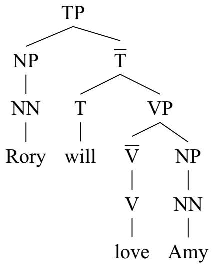
This rule allows us to qualify the sentence *“the book with a blue cover of poems” as being agrammatical, not only in English but in many other languages: *“le livre avec la couverture bleue de poèmes”$_{\mathrm{FR}}$ (French), *“leabhar na ndánta leis an gclúdach gorm”$_{\mathrm{IE}}$ (Irish), *“τὸ βιβλίο μὲ τὸ μπλὲ ἐξώφυλλο ποιημάτων”$_{\mathrm{EL}}$ (Greek), *“сборник с синей обложкой стихов”$_{\mathrm{RU}}$ (Russian), *“كتاب مع القصائد الأزرق الغطاء”$_{\mathrm{AR}}$ (Arabic), etc. It is harder to build a similar example in German since, in this language, “book

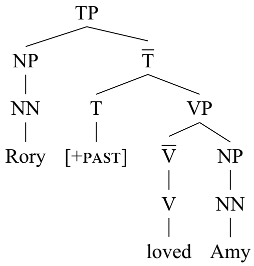
[4] Documents not typeset in TEX have a hard time adding the upper bar, probably because they use
☞ 346 only precomposed Unicode characters, and besides Ā, Ǣ, Ē, Ī, Ō, Ū, and Ȳ, no other uppercase Latin letters with upper bar exist in Unicode. In such documents, a prime is used instead of the upper bar (N′, V′, etc.), but the theory is still called “X-bar theory.”

of poems” is a compound word (“Gedichtbuch”$_{\text{DE}}$). Therefore we cannot break it to insert the information on the book’s cover: the complement is as close to the noun as it can get.

5.6.1 Tense Phrases

In p. 101, we mentioned the “TP” tag, and now we have the means to describe it. We know that auxiliary verbs (“be,” “have,” “do,” “will,” “can,” “shall,” etc.) combine with verbs to indicate tense, aspect, modality, etc. In $\overline{\text{X}}$ theory, there is a particular category for them, called T (as in “tense”). This category is also used to carry the morphological information of tense: ([-PAST] for present tense, and [+PAST] for past tense). To use T, we also need $\overline{\text{T}}$, and in that case, we use TP (“tense phrase”) as the tag of the root element instead of S. Here are two examples of tense phrases: “Rory will love Amy” and “Rory loved Amy”:





This concludes the part on Chomsky’s grammars and their various extensions. In the following, we will describe a radically different way of dealing with constituents.

5.7 Head-Driven Phrase Structure Grammars

Unlike Chomsky’s theories that somehow focus on syntax only, the theory of *Head-Driven Phrase Structure Grammars* (HPSG) is holistic: it deals with constituents, but also with words, agreement, complements, binding, etc. The various types of information are aggregated into a new kind of elementary object, called a *feature structure*. The form this object is represented in is called an *Attribute-Value Matrix* (AVM). In its simplest form, it is just a tree of entries containing other entries, and so on, every entry consisting of a tag (the “attribute”) and a value. For example [52, p. 20], the word “she” will be described as:

$$\begin{bmatrix} \textsc{phon} & \text{/ʃi/} \\ \textsc{graph} & \langle \text{she} \rangle \\ \textsc{synsem} & \begin{bmatrix} \textsc{local} & \begin{bmatrix} \textsc{category} & \begin{bmatrix} \textsc{head}\ \textit{noun} & \begin{bmatrix} \textsc{case}\ \textit{nom} \end{bmatrix} \\ \textsc{subcat} & \langle\rangle \end{bmatrix} \\ \textsc{content} & \begin{bmatrix} \textsc{index}\ \boxed{1} & \begin{bmatrix} \textsc{per} & \textit{3rd} \\ \textsc{num} & \textit{sing} \\ \textsc{gend} & \textit{fem} \end{bmatrix} \\ \textsc{restr}\ () & \end{bmatrix} \\ \textsc{context} & \begin{bmatrix} \textsc{backgr} & \left\{ \begin{bmatrix} \textsc{reln} & \textit{female} \\ \textsc{inst} & \boxed{1} \end{bmatrix} \right\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(In this example, we added the GRAPH attribute, which the original HPSG documentation omits, and Sproat [60, p. 10] calls ORTH). The SYNSEM attribute contains the word's syntactic/semantic contribution. The LOCAL attribute is about the local node of the syntax (as opposed to a NONLOCAL attribute containing information on gaps, which are empty nodes, very similar to those of transformational grammar, etc.). The CATEGORY attribute contains not only the syntactic category of the word but also the arguments it requires, which appear in the SUBCAT attribute. The CONTENT attribute contains the semantic interpretation of the word, and indices INDEX are reference markers, that is, connections between different locations of the feature structure.

It is essential to distinguish CONTENT and CONTEXT. The former represents contributions to the literal meaning, while the latter corresponds to presuppositions or implicatures. In our case, we have a background presupposition that the referent of "she" must be *female* (as opposed to the gender of "she" being *feminine*). The attribute INST signifies that the specific background information relates to the meaning $\boxed{1}$ of "she" (one can imagine polysemic words having different presuppositions attached to different word meanings $\boxed{1}$, $\boxed{2}$, etc.). From a structural point of view, it is interesting to note that an AVM ceases to be a tree when we consider indices $\boxed{1}$, $\boxed{2}$, … and instances pointing to them.

Morphological phenomena are described by *rules* involving feature structures. As an example, let us take the plural of nouns. We have the following rewrite rule:

$$\begin{bmatrix} \textsc{phon} & \boxed{1} \\ \textsc{graph} & \boxed{2} \\ \textsc{head} & \textit{noun} \\ \textsc{spr} & \langle \textit{count}+ \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \textsc{phon} & f\left(\boxed{1}\right) \\ \textsc{graph} & g\left(\boxed{2}\right) \\ \textsc{num} & \textit{plur} \end{bmatrix}$$

This rule is applied to countable nouns (SPR stands for "specifier"). It maps the input (of phonological representation $\boxed{1}$ and graphemic representation $\boxed{2}$) to an output feature structure with phonological representation $f(\boxed{1})$, graphemic representation $g(\boxed{1})$, and plural number. Notice that attributes not present in the rule remain as they are (this is *feature unification*) and that in a rule, we don't need to write complete paths of attributes, provided there is no ambiguity. In the case of the word "cat," function f will consist in appending the phoneme /s/ (so that /kæt/ becomes /kæts/), and function g will consist in appending the grapheme <s>.

Feature structures are also used to write syntax rules. For that, we apply two general principles:

1. the *Head-Complement Rule*: when we have a node the feature structure of which has an attribute COMPS ("complements") with n complements, and this node is followed by n siblings, then these $n+1$ nodes are the right-hand part of a rewrite rule. The left-hand part of the rule is a phrase node with empty COMPS attribute:

$$\begin{bmatrix} \text{phrase} & \\ \textsc{comps} & \langle\,\rangle \\ \textsc{head} & \boxed{0} \end{bmatrix} \rightarrow \boxed{0}\begin{bmatrix} \text{word} & \\ \textsc{comps} & \langle \boxed{1}, \boxed{2}, \ldots, \boxed{n} \rangle \end{bmatrix} \quad \boxed{1} \quad \boxed{2} \quad \cdots \quad \boxed{n}$$

2. the *Head-Specifier Rule*: when we have a node the feature structure of which has an attribute SPR ("specifier," in $\overline{\text{X}}$ theory it is the node preceding an X node under a $\overline{\text{X}}$ node) of value $\boxed{1}$ and is preceded by a node of type $\boxed{1}$, then these two nodes are the right-hand part of a rewrite rule. The left-hand part of the rule is a phrase node with empty SPR attribute:

$$\begin{bmatrix} \text{phrase} & \\ \textsc{head} & \boxed{2} \\ \textsc{spr} & \langle\,\rangle \end{bmatrix} \rightarrow \boxed{1} \quad \boxed{2}\begin{bmatrix} \textsc{spr} & \langle \boxed{1} \rangle \\ \textsc{comps} & \langle\,\rangle \end{bmatrix}$$

Using these two principles, we can build valid syntax trees. Let us take the sentence, "Amy speaks to Rory." In the feature structure of "speaks," we will have an attribute SPR of value NP and an attribute COMPS of value $\langle \text{PP}[\text{to}] \rangle$ (a prepositional phrase with the preposition "to"). Also, in the feature structure of "to," we will have an attribute COMPS of value $\langle \text{NP} \rangle$. The result can be seen in Fig. 5.1. As we see, for nodes with a single child, the head is the child. But whenever we have two children, we apply either the head-complement rule (for PP and VP) or the head-specifier rule (for S). In the former case, the head is the left node, and in the latter case, the right node.

To conclude, we observe that HPSG grammars introduce lexical information into nodes: we consider that the verb "speaks" takes an NP specifier and a PP complement, with the preposition "to." Without these elements, the syntax tree cannot be built. This will also be the case in the next section, where we will deal with combinatory categorial grammars: once again, lexical information will be indispensable to building valid syntax trees.

5.8 Combinatory Categorial Grammars

Let us return to the point where we have defined constituents, and we were looking for a way to obtain them algorithmically. Our first attempt was the use of context-free formal grammars. As we saw, starting with a noun, we have built nodes called NP (noun phrase) as well as nodes called $\overline{\text{N}}$ (intermediate noun-bar node), the choice among them depending on the sentence. Also, we have seen that context-free formal grammars need additional rewrite rules to handle phenomena such as verb valency, agreement, etc., at the risk of combinatorial explosion.

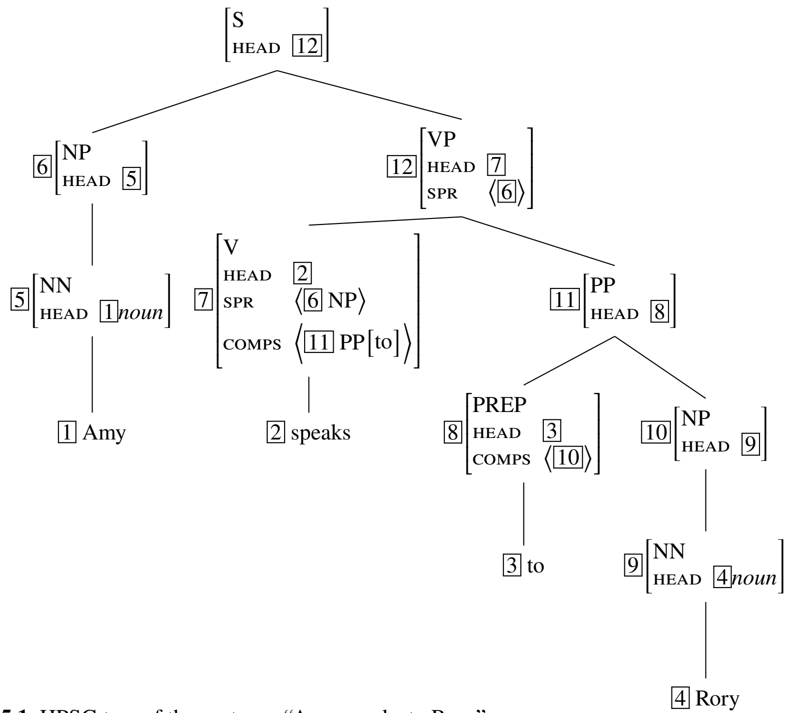
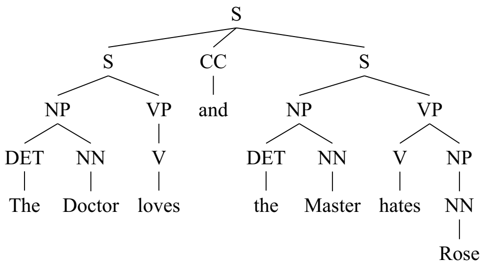


Fig. 5.1 HPSG tree of the sentence “Amy speaks to Rory”

In the following, we will purposely use a sentence not handled by previously described methods: “The Doctor loves and the Master hates Rose.” This sentence has two subjects and two verbs, with a common direct object. Here is the syntax tree produced by the online service `corenlp.run` of Stanford University:



It is not hard to see that this interpretation of the sentence is *wrong*: the verb “loves” is used as if it were intransitive, and “Rose” is presented as if it were the direct object of “hates” only, and not of “loves.” This is not the fault of `corenlp.run`, but is rather due to the fact that context-free grammars cannot model a situation where two verbs have a common direct object.

The method we will describe in this section is called *combinatory categorial grammars* [61] and was introduced by Steedman in the 1980s. It is based on two ingenious ideas:

1. instead of using generic POS tags, one uses tags that describe the behavior of each word (this idea originates from Ajdukiewicz's categorial grammars, introduced in 1935 [1]);
2. these tags are then used to calculate tags of constituents algorithmically, moving up the tree until we get the tag S of the =.

For example, in the sentence "Amy sleeps," the POS tag of "Amy" is NP, and the POS tag of "sleeps" is not V but S\NP. Here, the backslash means that we expect a constituent on the left side (a slash would be used for the right side); the NP in S\NP means that we expect an NP on the left of the current constituent, and the S in S\NP means that the constituent obtained when S\NP is combined with NP will be of type S.

This means that tags carry lexical information and that this information can be used in various ways to construct valid syntax trees: "sleeps" being intransitive, we will use tag S\NP (we call it a *category*); a transitive verb such as "loves" will instead use category (S\NP)/NP. For a verb with variable valency, we will be able to try out both categories and see which one allows us to build a valid syntax tree.

But the real innovation of Steedman's CCGs is the fact that categories are considered as objects of *combinatory calculus* [33, 3]. One can apply *combinators* to these objects to obtain new objects.

A *combinator* is an operation on functions. As we recall from high school, when we apply a function (e.g., sin) to a value (e.g., $\frac{\pi}{2}$), we get a new value (e.g., 1). This operation is called *application*. The situation here is a bit different: the functions on which combinators are acting live in a space called *combinatory logic* (CL) [3]. In this space, a function is applied not to a value but to another function, and what results from the application is yet another function. What matters is the order in which functions are applied. In CL, fgh means "f is applied to g, and the result of this application is applied to h," while $f(gh)$ means "we first apply g to h and then f to the result of this application."

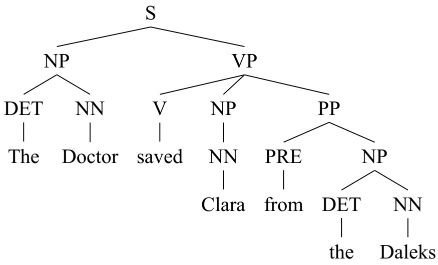
Here are some combinators: **I** is the identity combinator: it will send a function to itself, i.e., $\mathbf{I}f \equiv f$. **K** will take two functions as arguments and forget the first: $\mathbf{K}fg \equiv f$. And **S** will take three arguments $\mathbf{S}fgh$ and combine them in the following way: $\mathbf{S}fgh \equiv (fh)(gh)$, we apply g to h to obtain s, we apply f to h to obtain r, and then we apply r to s.

Combinators will allow us to model special cases that CFGs cannot handle. And we will never be short of combinators. Indeed, combinatory calculus provides us with an infinite number of combinators, and all combinators can be created by combining the three elementary combinators **I**, **K**, and **S**.

Let us now see how to convert CFGs to categorial grammars and then how to add combinators.

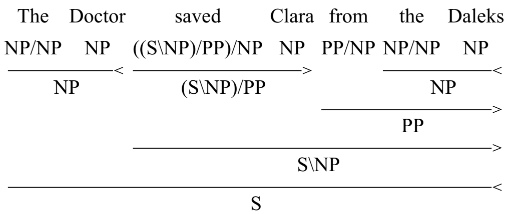
5.8.1 From Phrase-Structure Grammars to Categorial Grammars

Let us take the syntax tree of page 98:



In the case of "The Doctor" and "the Daleks," we have NPs that are combined on the left with a determiner to produce again NPs—we will give determiners the category NP/NP (as in "give me an NP and I will return a NP"). Similarly, we will not label "from" by a tag PRE as we did in the syntax tree above since we need a PP out of the combination of "from" and "the Daleks." Therefore, "from" will take the category PP/NP ("give me an NP on my right, and I will return a PP"). By writing everything down, there is only one possible choice for "saved," namely ((S\NP)/PP)/NP ("give me first an NP on my right, then a PP on my right, and finally an NP on my left, I will return an S").

In our notation, we place syntaxs on top and then go downwards until obtaining S, so we have leaves on the top and the root at the bottom, like in real-life trees. Notice the > or < symbol at the right of each horizontal line. It denotes the kind of application used ("forward" for > and "backward" for <).



Any syntax tree can be rewritten as a categorial grammar tree—for the moment, the only added value is that words get categories describing their behavior. For example, for any sentence in which the verb "to save" is used as "X saves Y from Z," we can use the category ((S\NP)/PP)/NP. The same verb used as "X saves Y" will have the category (S\NP)/NP, and an intransitive verb will have the category S\NP.

But Steedman [62] goes further: by using *combinators* at the derivation steps, he manages to handle challenging or even impossible cases for CFGs.

5.8.2 Conjunction

A conjunction at the level of nouns (as in "Amy and Rory") is not a problem. But what about a conjunction of verbs with the same subject and the same direct object? Let us take the sentence, "Daleks conquer and exterminate humans." The POS tag of the word “and” is CONJ (conjunction). We would use (S\NP)/NP if we had a single verb. We, therefore, need an operator that will merge the two verb categories into a single (S\NP)/NP category. This is the role of the *conjunction combinator* **Φ**. It replaces an X CONJ X category (for arbitrary X) by X. Here is the corresponding derivation:

$$
\begin{array}{cccccl}
\text{Daleks} & \text{conquer} & \text{and} & \text{exterminate} & \text{humans} & \\
\mathrm{NP} & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} & \mathrm{CONJ} & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} & \mathrm{NP} & \\
\cline{2-4}
 & \multicolumn{3}{c}{(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}} & & {<}\Phi{>} \\
\cline{2-5}
 & \multicolumn{4}{c}{\mathrm{S}\backslash\mathrm{NP}} & {>} \\
\cline{1-5}
\multicolumn{5}{c}{\mathrm{S}} & {<}
\end{array}
$$

5.8.3 Composition, Bluebird

Let us amend our example: “Daleks conquer and want to exterminate humans.” As conjunction can only be applied to identical categories, the category of “want to exterminate” must be identical to the one of “conquer,” which is (S\NP)/NP. As “want” is followed by a VP, it should logically have a category (S\NP)/VP. Also, “to exterminate” should take an NP (the direct object) and return a VP since the category of “want” expects a VP. But if “want” is (S\NP)/NP and “to exterminate” is VP/NP, how can their combination be (S\NP)/NP?

Here Steedman [62, pp. 40–43] introduces a new combinator **B** called *forward composition* or *bluebird* (the bird name comes from the fable *To Mock a Mocking Bird* by Richard Smullyan [57, 26], in which combinators are birds in an enchanted forest). This combinator works like the composition of mathematical functions: if f and g are functions, then $f \circ g$ is the function that sends x to $f(g(x))$, and this is exactly what **B** does: $\mathbf{B} f g x = f(gx)$. Combinator **B** is applied to a pattern X/Y Y/Z of categories and returns X/Z, i.e., it eliminates the common part Y.

If “want” is (S\NP)/VP and “to exterminate” is VP/NP, then “want to exterminate” will have the category (S\NP)/NP, as expected.

But what about “to”? If “exterminate” is a verb that takes an NP as a direct object, then its category should be (S\NP)/NP. What should be the category of “to” to obtain VP/NP when combined with (S\NP)/NP? We will, once again, use the forward composition combinator and define the category of “to” as being VP/(S\NP). Here is the derivation diagram:

$$
\begin{array}{cccccccl}
\text{Daleks} & \text{conquer} & \text{and} & \text{want} & \text{to} & \text{exterminate} & \text{humans} & \\
\mathrm{NP} & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} & \mathrm{CONJ} & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{VP} & \mathrm{VP}/(\mathrm{S}\backslash\mathrm{NP}) & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} & \mathrm{NP} & \\
\cline{5-6}
 & & & & \multicolumn{2}{c}{\mathrm{VP}/\mathrm{NP}} & & {>}\mathbf{B} \\
\cline{4-6}
 & & & \multicolumn{3}{c}{(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}} & & {>}\mathbf{B} \\
\cline{2-6}
 & \multicolumn{5}{c}{(\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP}} & & {<}\Phi{>} \\
\cline{2-7}
 & \multicolumn{6}{c}{\mathrm{S}\backslash\mathrm{NP}} & {>} \\
\cline{1-7}
\multicolumn{7}{c}{\mathrm{S}} & {<}
\end{array}
$$

In case the reader is wondering, there is also a *backward composition* combinator, also denoted by **B**, that takes categories Y\Z and X\Y and returns X\Z [62, p. 41].

5.8.4 Type Raising, Thrush

Let us now consider the sentence at the beginning of this section, namely, "The Doctor loves and the Master hates Rose." Let us first consider the part "The Doctor loves and the Master hates": its category must be S/NP, so that combined with the NP of "Rose," we get an S. As conjunction is applied to identical categories X and returns X, we know that both "The Doctor loves" and "the Master hates" must be of category S/NP.

The verbs "loves" and "hates" are transitive and are applied to an NP, so their categories must be (S\NP)/NP. And, as we have seen before, "The Doctor" is an NP, and therefore, "the Master" is also an NP. How can we combine the NP of "The Doctor" with the (S\NP)/NP of "loves" to obtain a S/NP?

☞ 170 The solution is to use a combinator **T** called *type raising* or *thrush* [62, p. 43]. **T** is a unary combinator defined as $\mathbf{T} f g \equiv g f$, i.e., it inverts the order of application. It will take an X and return T/(T\X) or T\(T/X), where T is some licensed category, for example S.

The expression S/(S\X) may seem intriguing. Intuitively, S\NP is a category seeking an NP on its left and returning a sentence S, i.e., S\NP is a sentence whose subject is missing (in English, the first NP of a sentence is often the subject). Therefore, S/(S\NP) means: "Give me a sentence without a subject, and I will return a complete sentence." In other words, a type-raised NP is not simply an NP but a sentence-completer, a category that completes sentences by adding the missing NP as a subject. Here is the derivation diagram:

```
 The    Doctor   loves       and    the    Master   hates       Rose
NP/NP    NP    (S\NP)/NP   CONJ   NP/NP    NP     (S\NP)/NP    NP
----------->                      ----------->
     NP                                NP
----------->T                     ----------->T
  S/(S\NP)                          S/(S\NP)
------------------------->B       ------------------------->B
          S/NP                              S/NP
-------------------------------------------------------------<Φ>
                            S/NP
---------------------------------------------------------------------->
                               S
```

By type-raising NPs, we get S/(S\NP)s, which we then compose with verbs and get two S/NP. Then, these two S/NP become entry points of the conjunction. The conjunction provides a single S/NP, which we apply to the unique object, "Rose," and the tree is complete. Notice that this diagram, contrarily to the phrase-structure syntax tree of p. 110, is *symmetric*: "Rose" is, equally importantly, the direct object of both verbs.

5.8.5 A Python Parser for CCGs

The Python *nltk* package [4] includes a CCG module[5] that allows parsing =s to build CCG syntax trees. There is no trained universal lexicon for English, so one has to provide the lexicon. Here is the code for parsing the sentence “The Doctor loves and the Master hates Rose,” including a minimal lexicon for this specific sentence:

```
1  from nltk.ccg import chart, lexicon
2  lex = lexicon.fromstring('''
3          :- S, NP
4  
5          Det :: NP/NP
6          Modal :: (S\\NP)/NP
7  
8          the => Det
9  
10         Doctor => NP
11         Master => NP
12         Rose => NP
13 
14         loves => Modal
15         hates => Modal
16 
17         and => var\\.,var/.,var
18 
19         ''')
20 parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
21 for parse in parser.parse("the Doctor loves and the Master\
22     hates Rose".split()):
23     chart.printCCGDerivation(parse)
24     break
```

On line 3 we define the basic categories we will use (i.e., `S`, `NP`). Then, on lines 5 and 6 we define some shortcuts that will not appear in the syntax tree (`Det` for determiners, `Modal` for verbs). On lines 8–15 we introduce the lexical category of each word, and, finally, on line 17, the category of the conjunction “and,” which uses the special symbol `var`. This symbol means that any category can be used, even of a very complex type (the dot prevents permutation, and the comma prevents composition). Here is the result we obtain:

```
   the    Doctor    loves               and                the    Master     hates     Rose
 (NP/NP)   NP    ((S\NP)/NP)  ((_var0\.,_var0)/.,_var0)  (NP/NP)   NP    ((S\NP)/NP)  NP
----------------->
       NP
----------------->T
   (S/(S\NP))
----------------------------->B
            (S/NP)
                                                       ----------------->
                                                              NP
                                                       ----------------->T
                                                          (S/(S\NP))
                                                       ----------------------------->B
                                                                   (S/NP)
```

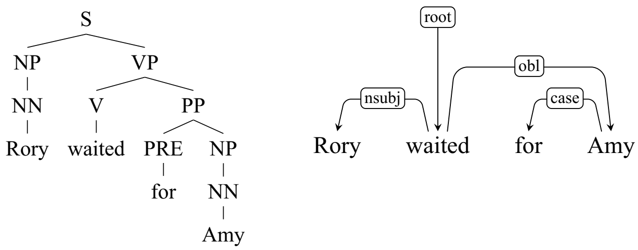
[5] https://www.nltk.org/api/nltk.ccg.html

```
                                 ----------------------------------------------------->
                                                   ((S/NP)\.,(S/NP))
-------------------------------------------------------------------------------------<
                                         (S/NP)
------------------------------------------------------------------------------------------->
                                               S
```

In this section, we have only scratched the surface of the CCG approach. We will continue with an approach that has become increasingly popular in computational linguistics in recent decades. This approach is called *dependency syntax*.

5.9 Dependency Syntax

Dependency syntax is another way of describing relations between words or word groups. It also uses syntax trees, but instead of adding new nodes for each constituent or intermediate symbol as in $\overline{\mathrm{X}}$ theory, the nodes of the dependency syntax tree are simply the words of the sentence. Some word (usually the verb) is chosen as being the *root* of the sentence, and all words are accessible from the root using directed paths. Edges are *dependencies*. In each dependency, the predecessor node is called *head* and the successor node, *dependent*. Dependencies correspond to syntax functions and are labeled as such. Here is an example of a phrase structure syntax tree (on the left) and dependency syntax tree (on the right) for the same sentence:



One of the significant advantages of dependency grammars becomes immediately visible when comparing the two trees above: in the dependency syntax tree, there is a strong connection between “waited” and its complement “Amy” since a single edge connects the two, while in the phrase structure tree, the connection is less visible. In NLP, this is a clear advantage: to give an example, in sentiment mining related to a given product, using dependency syntax trees, one can evaluate the sentiment polarity of a verb (and its potential adverbial modifiers) and then check nodes at distance 1 from the verb to find whether the product one is looking for is a complement of the polarized verb.

DEDICACE

A mes enfants, Michel, Bernard et Yveline Tesnière, je dédie ce livre dont leurs curiosités d'élèves de sixième A ont hâté la maturation.

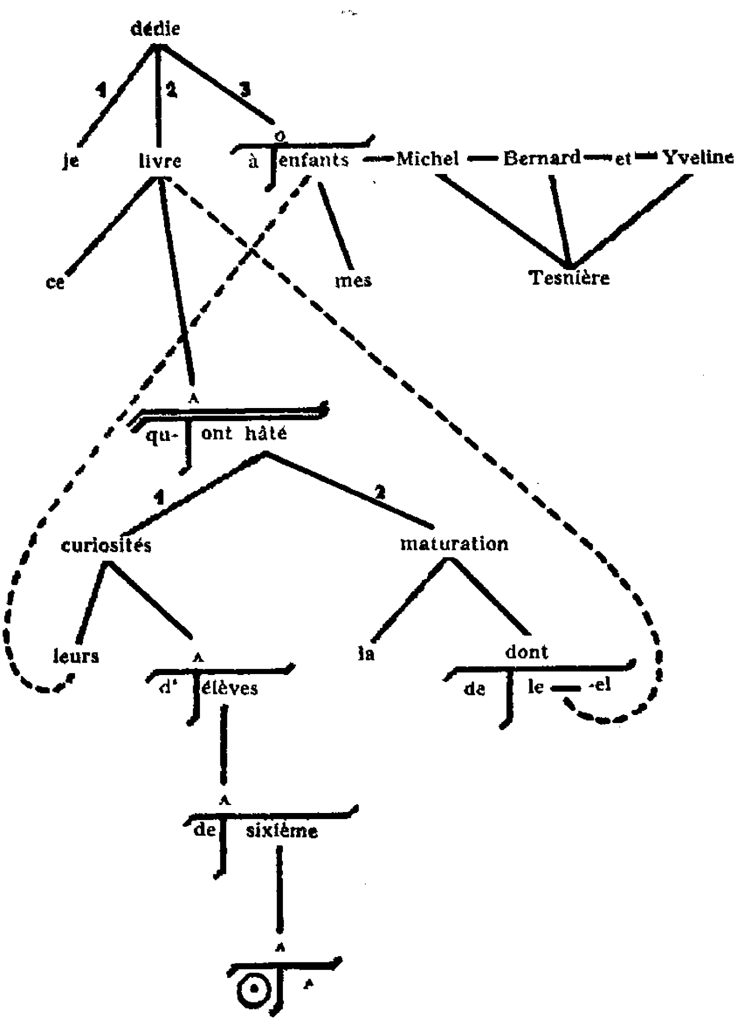


Fig. 5.2 "To my children, Michel, Bernard, and Yveline Tesnière, I dedicate this book whose maturation was hastened by their curiosity as sixth A graders." Dedication [64, p. v] of Lucien Tesnière to his children, followed by dependency syntax graph of the dedication, in his original notation. Dotted lines denote anaphoras. The T-shape symbol denotes a phenomenon that Tesnière calls *transfer* [64, pp. 364–371]: a prepositional phrase acts like an adjective (therefore the A on top of the symbol). The circled dot symbol denotes a translation without preposition [64, p. 407], namely the one of the "A" modifier. At that time, in the French education system, there were two sixth-grade classes, called "sixth A" ("classical") and "sixth B" ("modern") so that in the sentence, "A" is a modifier of the adjective "sixth." The sentence is complex on purpose to illustrate as many phenomena as possible.

5.9.1 Some History

> [M]any books on syntax and grammar that have appeared in the last 60 years in the English-speaking world have taken phrase structure for granted, that is, they are seemingly unaware of the alternative approach to syntax that dependency brings to the table.
>
> Osborne, *A Dependency Grammar of English*

In 1959, that was two years after Chomsky's first major opus, was published Lucien Tesnière's *Éléments de syntaxe structurale* [64, 65], a posthumous work shedding new light on syntax relations between words. A year later, David Hays, an American linguist at RAND Corporation in California, presented his approach to dependency syntax at a Symposium on Machine Translation [29] at UCLA. Historians claim (e.g., [47, p. 74]) that Hays was influenced by Tesnière, but neither in the UCLA text nor in a subsequent article in the prestigious journal *Language* [28], does Hays mention Tesnière. He instead refers to Soviet works as his source of inspiration.

Whether Hays was inspired by Tesnière or not, what is essential is the fact that Hays was very influential in Natural Language Processing. He even wrote the first book in this field in 1967: *Introduction to Computational Linguistics* [31]. For this reason, dependency syntax has been considered as a "computational" approach from the very beginning, so most textbooks on (theoretical) syntax ignore it [51, p. 34]. Notwithstanding this fact, the success of dependency syntax in Natural Language Processing has been overwhelming. This section will give some theoretical background on dependency syntax, mainly inspired by Osborne [51].

5.9.2 Strings and Catenae

To explore dependency grammar, we need some new vocabulary [51, p. 107]:

string: a word or a sequence of words that are adjacent for the linear order of the sentence;

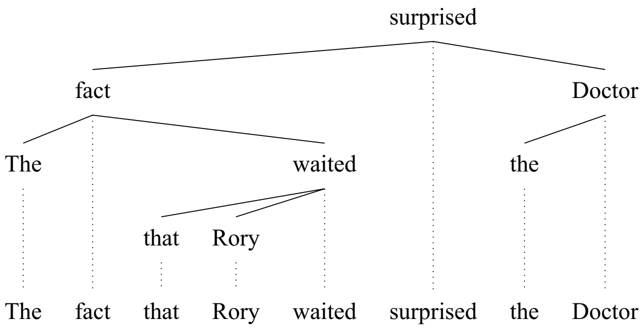
catena (the Latin word for "chain," the plural being "catenae"): a word or a sequence of words that belong to an (undirected) subtree of the dependency syntax tree. Notice that the order we use for this sequence of words is the linear order of the sentence and not the order induced by paths in the tree;

component: a word or a sequence of words that is both a string and a catena;

complete subtree: a subtree T' of a tree T is *complete* if there is a node $V \in T'$ such that T' is generated by all paths leaving V;

constituent: a component that is a complete subtree.

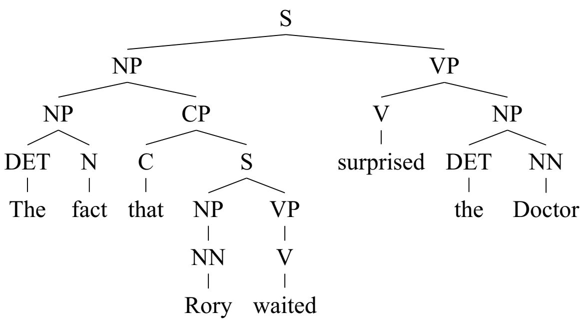
Intuitively, a string is a "horizontal" structure. At the same time, a catena is a "vertical" structure, a component is both, and a constituent is a component related to a specific tree node. Let us take an example to illustrate these notions:



The notation we use is the one of Osborne [51]. It allows a good understanding of the difference between string and catena. Note that even though we do not draw arrowheads, the tree is nevertheless directed (from top to bottom) and rooted (the root, in our example, being “surprised”). A string is any sequence of adjacent words in the sentence, e.g., (“fact,” “that,” “Rory”), or (“surprised,” “the”), or the whole sentence, or any single word. Examples of catenae are: (“fact,” “that,” “waited”) or (“surprised,” “the,” “Doctor”).

Examples of node sets that are not catenae are: (“fact,” “Doctor”) because any subtree of the syntax tree containing these nodes necessarily also contains the node “surprised” by connectivity, so the two nodes do not constitute a subtree, or (“waited,” “surprised”) (for the same reason). We remark that (“waited,” “surprised”) is a string but (“waited,” “surprised”) is not a catena and, inversely, (“fact,” “waited”) is a catena but (“fact,” “waited”) is not a string.

Here are some components: “fact that Rory waited surprised,” “surprised the Doctor,” and “Rory waited.” The first one is not a constituent since it is incomplete—it becomes complete when we add the word “The”: “The fact that Rory waited.” The second one is not a constituent: it contains the root, and therefore, the only complete subtree containing it can only be the entire tree. The third one is not complete either. To complete it, we must add “that”: “that Rory waited.” “The fact that Rory waited” and “that Rory waited” are both constituents, the types of which we encountered in the phrase-structure grammar approach: the first is the NP of the global sentence and the second one a subordinate clause CP, as can be seen in the phrase-structure tree of the sentence:



5.9.3 Types of Dependency Relations

As described in Osborne [51], dependency grammar is based on binary relations between head and dependent only, without any extra information about the nature of the relation. The Universal Dependencies (UD) annotation framework is an initiative for providing consistent and relevant annotations for dependency relations on an international level: as of October 2023, 141 languages are covered (with 66 possible future extensions), together with freely available annotated corpora of various sizes (ranging from corpora with more than 100k sentences in Japanese, Czech, and Russian, to corpora with ten, eight or even only a single sentence. The single-sentence corpus, likely to enter the Guinness Book of Records, is for the Neapolitan language, an Italo-Dalmatian language spoken in Southern Italy. Here is the unique sentence: "A uaglionă scrivettĕ 'na lettĕră all' a la amica sojă"$_{\text{NEA}}$ ⌊the girl wrote a letter to her friend⌋).

UD defines 37 syntactic relations that cover all languages (subtypes are allowed but not standardized). Here is the complete list taken from de Marneffe *et al.* [48, p. 266], together with occurrence frequencies in the English UD corpora and examples (the order is loosely frequency-based):

`punct` (11.05%[6]): punctuation attached to the head of its clause or phrase. The head of the clause or phrase is also the head of this relation, the punctuation mark being the dependent;
`case` (10.53%): links a case-marking element (preposition, postposition, or clitic) to a nominal, e.g., "Doctor" → "by" in "Clara was saved by the Doctor";
`det` (8.19%): determiner (article, demonstrative, etc.) in a nominal, e.g., "Doctor" → "the" in the previous example;
`nsubj` (7.27%): nominal subject; nominal core argument which is the syntactic subject (or pivot) of a predicate, e.g., "created" → "Davros" in "Davros created the Daleks";
`nsubj:pass` (0.61%): a particular case of `nsubj`, when the verb is in passive voice, e.g., "saved" → "Clara" in "Clara was saved by the Doctor";
`root` (6.05%): the root of the sentence, e.g., "saved" in the previous example;
`nmod` (5.17%): nominal modifier; a nominal modifying another nominal, e.g., "circuit" → "Tardis" in "the translation circuit of the Tardis";
`nmod:poss` (1.59%): a particular case of the previous when a possessive pronoun is used, e.g., "circuit" → "Tardis" in "the Tardis's translation circuit"; but also
`compound` (3.19%): any word-level compounding (noun compound, serial verb, phrasal verb), e.g., "circuit" → "translation" and "translation" → "Tardis" in "the Tardis translation circuit" (as a term, all words in the nominative case); there is also
`flat` (1.28%): flat multiword expression; links elements of headless semi-fixed multiword expressions like names, e.g., "Winston" → "Churchill" in "Win-

[6] Meaning that 11.05% of the 572,795 dependency relations of the merged English corpora of the UD project are of type `punct`. As of May 2023, the merged corpora contain 45,982 sentences.

ston Churchill" (most NLP parsers will annotate this by `compound` with the head/dependent relation reversed);

`amod` (4.93%): adjectival modifier of a nominal, e.g., "wolf" → "bad" in "bad wolf";

`obl` (4.81%): oblique; a nominal functioning as a modifier of a predicate that is neither a subject nor a direct object or an indirect object, e.g., "opened" → "screwdriver" in "he opened the door with the sonic screwdriver," taking the preposition "with" into account, not to be confused with

`iobj` (0.32%): indirect object; nominal core argument of a verb that is not its subject or (direct) object, e.g., "told" → "Amy" in "Madame Kovarian told Amy about her baby," since "Amy" is neither subject nor direct object, and there is no preposition that would make it an oblique dependency;

`obj` (4.72%): (direct) object; the core argument nominal, which is the most essential core argument that is not the subject, typically the most directly affected participant, e.g., "created" → "Daleks" in "Davros created the Daleks";

`advmod` (4.49%): adverb or adverbial phrase modifying a predicate or modifier word, e.g., "solved" → "brilliantly" in "he solved it brilliantly";

`conj` (3.52%) conjunct and `cc` (3.08%); the former links two elements that are conjoined, and the latter links a coordinating conjunction to the following conjunct. Here are some examples

- "Adric" $\xrightarrow{\texttt{conj}}$ "Nyssa" and "Adric" $\xrightarrow{\texttt{cc}}$ "and" in "Adric and Nyssa were companions" (conjunction between nous),
- "converted" $\xrightarrow{\texttt{conj}}$ "exterminated" and "converted" $\xrightarrow{\texttt{cc}}$ "or" in "Humans were converted or exterminated",
- "before" $\xrightarrow{\texttt{conj}}$ "after" and "before" $\xrightarrow{\texttt{cc}}$ "or" in "We can meet before or after dinner."

Note that because of the asymmetric nature of dependency relations (there has to be a head and a dependent), the UD project decided to take the first coordination element as head and all others as dependents. There is also a particular case of coordination without any conjunction particle, namely,

`parataxis` (0.58%): links constituents placed side by side with no explicit coordination or subordination, e.g., "saw" → "leaving" in "I saw her, she was leaving";

`mark` (3.22%): marker; links a function word marking a clause as subordinate to the predicate of the clause, e.g., "vanished" → "as" in "she wept as the Tardis vanished"; in the same context we also have

`advcl` (1.57%): adverbial clause modifying a predicate or modifier word, e.g., "wept" → "vanished" in the same example; and also

`acl` (0.83%): adnominal clause; a clause modifying a nominal, e.g., "fact" → "waited" in "the fact that Rory waited," where "Rory waited" is a clause modifying "fact";

`aux` (2.69%): auxiliary; links a function word expressing tense, mood, aspect, voice, or evidentiality to a predicate, e.g., "fighting" to "is" in "She is fighting Sillurians";

`aux:pass` (0.69%): a subtype of the previous when the verb is in passive voice, e.g., “abducted” to “was” in “Ashildr was abducted by the Mire”;

`cop` (1.87%): copula; links a function word used to connect a subject and a nonverbal predicate to the nonverbal predicate, e.g., “Doctor” → “am” in “I am the Doctor.” In English, the *copula* (plural: copulae) is the verb “to be” when it connects a subject with a nonverbal object. Still, in other languages (like Arabic or Russian), copulae can be missing or suffixes like “です”$_{\text{JA}}$ in Japanese “私はドクターです”$_{\text{JA}}$ ⌊I-(topic marker)-doctor-am⌋.

`xcomp` (1.33%) and `ccomp` (0.80%): clausal complements of a verb or adjective, i.e., a subordinate clause that is the object of the main verb. The difference between `ccomp` is that the subject of the subordinate clause can be arbitrary, while in `xcomp`, it has to be the same as the one of the main clause. The example given on the UD project’s Web site is the sentence “The boss said to start digging,” where we have the main verb “said” that takes “to start” as an object, and then “to start” takes “digging” as an object. The first dependency “said” → “start” is an *open clausal complement* since the boss (subject of “said”) is probably not the one to be digging, while the second dependency “start” → “digging” is an `xcomp` since the ones that will start are precisely those that will digg. In this case, we say that we have *subject control*, a notion that goes back to Chomsky’s Government and Binding Theory;

`csubj` (0.01%): clausal syntactic subject of a predicate, this is the case where the subject of our sentence is a clause, e.g., “made” → “said” in “What the Dalek said made no sense” where “What the Dalek said” is the subordinate clause that becomes the subject of “made”;

`nummod` (1%): numeric modifier; numeral in a nominal, e.g., “Doctors” → “five” in “the five Doctors”; also the case of

`clf`: (numeral) classifier; a word reflecting a conceptual classification of nouns linked to a numeric modifier or determiner. This happens, e.g., in Chinese, as in “五”$_{\text{ZH}}$ → “位”$_{\text{ZH}}$ in “五位医生”$_{\text{ZH}}$ ⌊five-(numeral classifier)-doctors⌋ (i.e., the number is the head and the numeral classifier the dependent) ;

`appos` (0.61%): appositional modifier; a nominal used to define, name, or describe the referent of a preceding nominal, e.g., “Truth” → “Sierra” in “Truth or Consequences, Sierra County, New Mexico”;

`expl` (0.3%): expletive; links a pronominal form in a core argument position but not assigned any semantic role to a predicate, typical examples are “rains” → “it” in “it rains” where the “it” does not refer to any entity (in French, the “il”$_{\text{FR}}$ of “il pleut”$_{\text{FR}}$ ⌊it rains⌋ is called an *impersonal pronoun*—in English, an example of impersonal pronoun is “one,” as in “one can never be too careful”);

`vocative` (0.06%): nominal directed to an addressee, e.g., “trust” → “Amy” in “Amy, trust me!”.

Follow some very rare cases that we can qualify as exotic:

`discourse` (0.36%): discourse element (interjection, filler, or non-adverbial discourse marker), e.g., “win” → “Wow” in “Wow, you win,” where “Wow” is a *cognitive interjection* representing surprise;

`fixed` (0.34%): fixed multiword expression; links elements of grammaticalized expressions that behave as function words or short adverbials, e.g., “as_1” $\rightarrow$ “as_2” and “as_1” $\rightarrow$ “well” in “Rose as well as Sarah Jane”;

`list`: links elements of comparable items interpreted as a list, e.g., “Donald” $\rightarrow$ “Stanford” in “Donald E. Knuth Stanford University”;

`reparandum`: repair of a (normally spoken language) disfluency, e.g., “bag” $\rightarrow$ “car” in the disfluent sentence “When you get on an airplane, your car- your [pause of 0.421s] bag is x-rayed”;

`goeswith`: links parts of a word that are separated but should go together according to standard orthography or linguistic wordhood, i.e., cases where a graphemic word separator should usually be absent or replaced by a hyphen, e.g., “never” $\rightarrow$ “more” in “Quoth the Raven nevermore”;

`orphan`: links orphaned dependents of an elided predicate, e.g., “you” $\rightarrow$ “coffee” in “I like tea and you coffee”;

`dislocated`: a peripheral (initial or final) nominal in a clause that does not fill a regular role of the predicate but has roles such as topic or afterthought. The example given in [48] is “like” $\rightarrow$ “Peter” in “Peter, I don’t like him”;

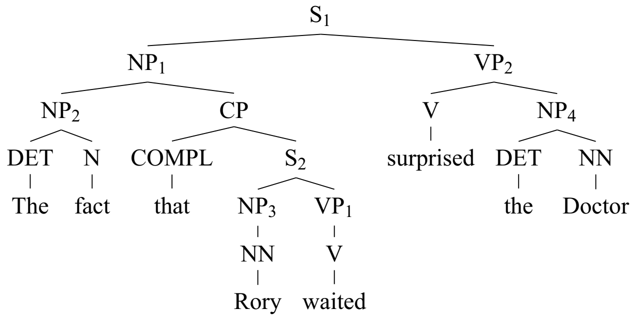
and finally, the case where the annotator is in the midst of a Kobayashi Maru situation (= a no-win scenario [18]) and nothing seems to fit:

`dep`: unspecified dependency, used when a more precise relation cannot be determined. We don’t give an example because this shouldn’t happen in the first place…

5.9.4 From Constituents to Dependencies

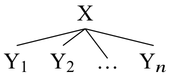
Let us see how to build a dependency tree once we know the constituents of our sentence.

Here is the phrase-structure syntax tree of the example of p. 119:



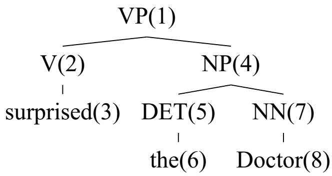
Let us convert it into a dependency tree.

First of all, in a phrase-structure syntax tree we distinguish two kinds of internal nodes: *endocentric* and *exocentric* [51, p. 48]. Endocentric nodes are those that carry a label obtained from one of their children, e.g., "fact" is an N and its parent is an NP, so NP_2 is endocentric; "surprised" is a V and its parent is a VP, so VP_2 is endocentric; the parent of NP_2 is an NP, so NP_1 is also endocentric; the parent of COMPL is a CP, so CP is endocentric, etc. On the other hand, S_1 and S_2 are necessarily exocentric since none of their children is an S.

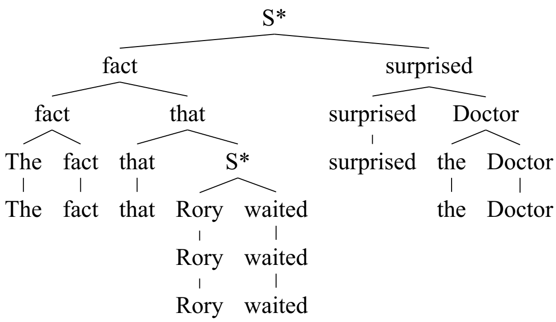
Secondly, as dependencies are binary relations, converting a phrase-structure node such as 

into a dependency will only work for $n = 2$ so that the resulting dependency is either $Y_1 \rightarrow Y_2$ or $Y_2 \rightarrow Y_1$. Having $n \leq 2$ children for each tree node means the tree is *binary*. Our example already fulfills this condition, but this is not the case of all trees. Nevertheless, when a phrase-structure tree hap-
☞ 105 pens not to be binary, we can always apply a method such as $\overline{\mathrm{X}}$ theory to convert it into a binary tree by adding bar-nodes [7, p. 168]. $\overline{\mathrm{X}}$ nodes are always endocentric since, by definition, they take their labels from one of their children.

Suppose we restrict ourselves to endocentric nodes and binary trees. In that case, converting a constituent tree into a dependency tree is straightforward. Among the two children of the node, the one that has the label that determines the parent's label will be the head and the other the dependent. For example, in

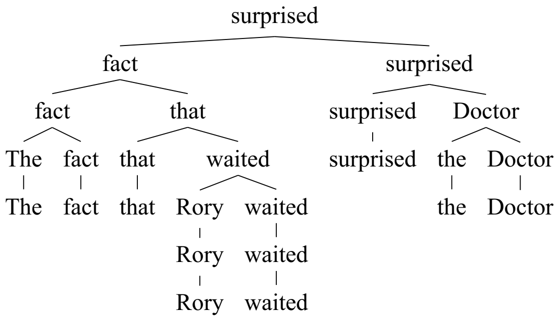


node (4) is an NP because node (7) is an NN; therefore, (7) is the head, and (5) is its dependent; the head (7) moves upwards and takes the place of node (4); then we have again an endocentric situation where (1) takes its label from (2), so that (2) is head and (4) dependent. To convert the phrase-structure tree into a dependency tree, we will move words upwards until there is a sibling, and we will replace every endocentric node with the head of its children. Here is the result, a hybrid phrase-structure/dependency grammar tree, where we have marked exocentric nodes by an asterisk:

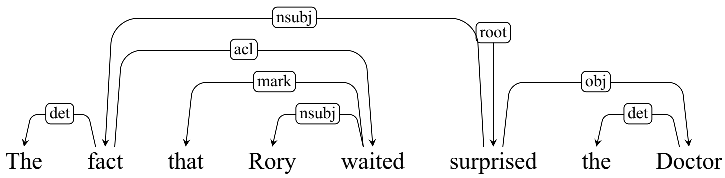


As we see, we have only two exocentric nodes. In both cases, they are S nodes with children NP and VP. The dependency relation between the two is not difficult to find:

it is `nsubj`, the "nominal subject" relation. As we can see on page 5.9.3, the head of this relation is the VP and the dependent is the NP. This allows us to complete the hybrid tree:



and from it, we can immediately extract the dependency tree. Adding the annotations from §5.9.3, we get:



where "Rory waited" is a subordinate clause that modifies the nominal "fact," the complementizer being "that."

5.9.5 Parsing Dependency Grammars in Python

There are many dependency parsers out there. Among those implemented in Python, there is *stanza* [55]. Here is the code for parsing the example sentence of the previous section:

```
import stanza
stanza.download('en')
nlp = stanza.Pipeline(lang='en', processors='tokenize,mwt,pos,lemma,depparse')
for text in ["The fact that Rory waited surprised the Doctor"]:
    doc=nlp(text)
    for sent in doc.sentences:
        for word in sent.words:
            print(word.id,word.text,word.lemma,word.upos,word.xpos,\
                word.feats,word.head,word.deprel)
```

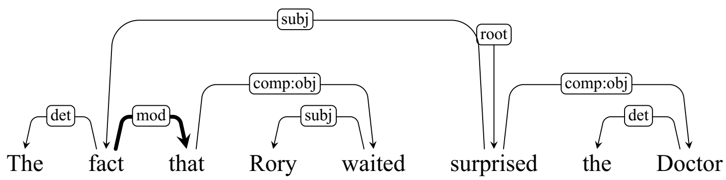
We have chosen the order of `word` object attributes in the `print` in such a way that the output is in CoNLL-U format:

```
1 The the DET DT Definite=Def|PronType=Art 2 det
2 fact fact NOUN NN Number=Sing 6 nsubj
```

```
3 that that SCONJ IN None 5 mark
4 Rory Rory PROPN NNP Number=Sing 5 nsubj
5 waited wait VERB VBD Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin 2 acl
6 surprised surprise VERB VBD Mood=Ind|Tense=Past|VerbForm=Fin 0 root
7 the the DET DT Definite=Def|PronType=Art 8 det
8 Doctor Doctor PROPN NNP Number=Sing 6 obj
```

5.9.6 Surface-Syntactic Universal Dependencies

In Sections 5.9.3 and 5.9.4, we considered dependency tags as defined by the project Universal Dependencies (UD). The reader should be aware of the fact that there is an alternative to UD, named SUD (Surface-Syntactic UD) [50, 36]. Contrary to UD, it is based on syntactic criteria (favoring functional heads), and the relations are defined on distributional and functional bases. Here is the dependency tree of the sentence "The fact that Rory waited surprised the Doctor," using SUD dependencies:



There is indeed a difference between this tree and the one of the UD scheme, on p. 125. We already mentioned that in this sentence, we have a subordinate clause that modifies the nominal "fact" and that "that" is its complementizer. In the SUD scheme, the complementizer depends on the nominal "fact" and not on the verb "waited." This is precisely what is meant by saying that SUD *favors functional heads*: here, "that" is the head of the subordinate clause, and "waited" is its verb—UD will favor the latter and SUD the former.

On the SUD Web site[7], as of October 2023, one can find eight corpora natively built in SUD, 243 corpora converted from the corresponding UD corpora and a conversion tool in both directions: from UD to SUD and from SUD to UD. The conversion tool is based on *grew*, a graph-rewrite tool [5] written in the OCaml language.

5.10 Psycholinguistic Aspects

Psycholinguistic theories of syntax adopt Chomsky's CFGs. According to a seminal paper by Frazier & Fodor [21], syntactic analysis by hearers or readers is performed in two steps: the first one called *Sausage Machine* (or *Preliminary Phrase Packager*, PPP) will assign lexical and phrasal nodes to groups of words; the second step

[7] https://surfacesyntacticud.github.io/

consists in adding higher nonterminal nodes to obtain a complete syntax tree, this is done by the *Sentence Structure Supervisor* (SSS).

The PPP is called a “sausage machine” because it will analyze a bunch (“sausage”) of 7 ± 2 words at a time, regardless of whether they form a clause, several clauses, or just part of a clause. The second step will then connect the sausages. Concerning memory issues, Frazier & Fodor [21] assert that the PPP is working with small sets of words to optimize memory usage, and, as for the SSS, even though it will have to deal with long (sometimes *very* long) sentences, this is no problem because there is an internal structure,

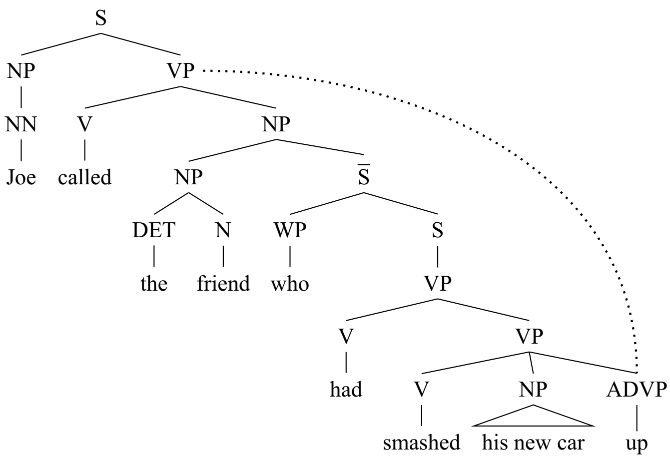
> for it is a well-attested (if unexplained) fact about human memory that the more structured the material to be stored, the smaller the demand it makes on storage space. [21, p. 293]

The sentences that exhibit the most problems of syntax analysis are *garden path sentences*, i.e., sentences that lead us to an initial interpretation that turns out to be incorrect when we arrive at the last word(s).

Three strategies help us process sentences, particularly garden path sentences, with optimal results. The first is the *right association principle*. The sentences

> Joe called the friend who had smashed his new car up.
>
> John read the note, the memo, and the letter to Mary.

are ambiguous: in the first one, the particle “up” may belong to “called” or to “smashed”; in the second, “to Mary” may be the complement of “read” or may be associated only with “the letter.” In both cases, we tend to choose the candidate closest to the part where ambiguity arises. The right association principle states that “terminal symbols optimally associate to the lowest nonterminal node.” Here is the syntax tree of the first example:



where the dotted line represents the other possible attachment of “up.” According to Frazier & Fodor [21, p. 297], the right association principle is so strong that it persists even when the sentence is unambiguous, and only the higher attachment is semantically possible, as in

Joe looked the friend who had smashed his new car up

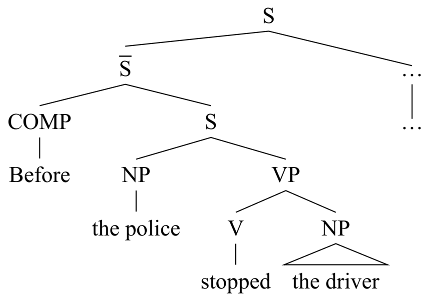
where “looked” would need an “at” particle if the “up” particle were attached to “smashed.” Despite this fact, we still connect “up” to “smashed” on a first pass.

The second strategy is the *late closure principle*, which states that when possible, we tend to attach material into the currently processed clause. Here is a sentence (from Warren [67, p. 167]) and its continuation:

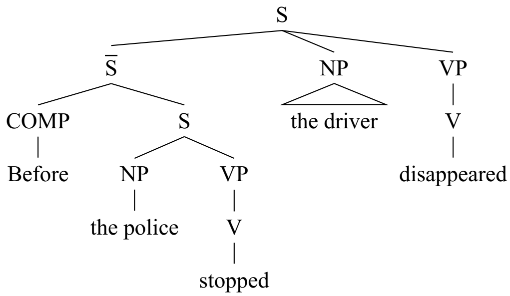
Before the police stopped the driver …

Before the police stopped the driver disappeared

For the first sentence, our sausage machine will provide the following tree:



and the node “the driver” will be attached to the closest VP. But when we process the last word, we have to reconsider the tree and replace it with the following:

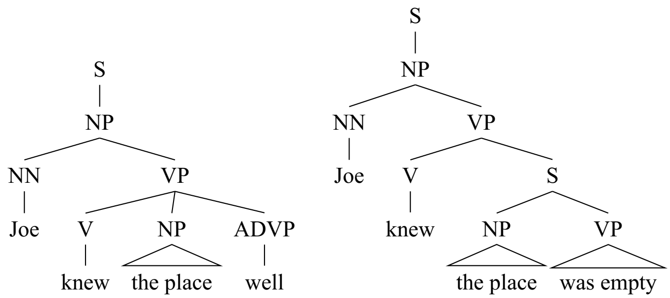


Finally, the *minimal attachment principle* states, “attach incoming material using the fewest possible nodes.” Consider the two sentences

Joe knew the place well

Joe knew the place was empty

Here are the tress of these sentences:

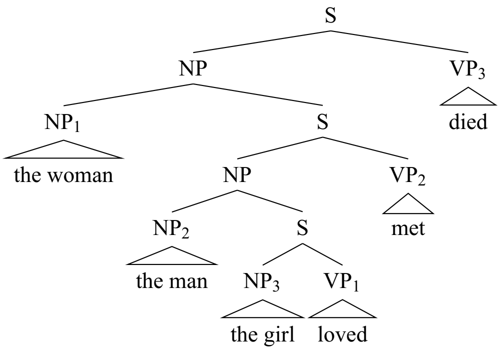


When we process the sentence and arrive at the word "place," we will probably have built the tree on the left because of the minimal attachment principle. Indeed, a hypothesis based on the second example would require an extra node S between the VP and the NP of "the place."

Frazier & Fodor [21] also explain why center-embedded sentences [15], such as

The woman the man the girl loved met died

are extremely difficult to parse. The difficulty of such sentences is the starting point of Watson's novel *The Embedding* (see [17]). According to Frazier & Fodor [21], the cause of the difficulty is the failure of the sausage machine to build meaningful sausages. Considering the tree



and the corresponding memory[8] requirements, Frazier & Fodor [21] conclude that, on the one hand, NP_1, VP_2, and VP_3 should be long enough to give rise to individual sausages, and, on the other hand, NP_2, NP_3, and VP_1 should be short. By using this approach, they rewrite the sentence as

[The very beautiful young woman]$_{NP_1}$ the man the girl loved [met on a cruise ship in Maine]$_{VP_2}$ [died of cholera in 1962]$_{VP_3}$

which is indeed miraculously easier to process despite the facts that it is still center-embedded, that it contains no explicit syntactic markers (such as "the woman *that*

[8] We purposely do not use the terms "working memory" or "short-term memory," since Frazier & Fodor do not use them in [21]. See Cowan [12] for the confusion between the two kinds of memory in the literature.

the man…") and no punctuation, and that it is, in fact, more than twice as long than the previous one.

5.11 Further Reading

5.11.1 Literature

> Why is syntax worth studying?
>
> Kuiper & Nokes, *Theories of syntax*

Kuiper & Nokes's *Theories of syntax* [40] is a textbook on syntax that gives a concise, clear, and pleasant-to-read overview of many theories. Carnie [7] is a good introduction to Chomskian syntax. Borsley & Börjars [6] is a good overview of non-transformational syntax theories, such as HPSG, CCG, or Lexical-Functional Grammar. Graffi's *200 Years of Syntax* [23] is a fascinating read for those interested in the history of syntax.

5.11.2 LaTeX

The best package to draw syntax trees and prosodic trees (as in Fig. 2.8) is *tikz-qtree*[9]. It takes over the syntax of *qtree* package[10], but embedded into the very powerful *tikz* graphics package.

For AVMs, there is a special package, *langsci-avm*[11], distributed by the publisher Language Science Press. To obtain the HPSG tree of Fig. 5.1, we used *tikz-qtree* with *avm* structures at nodes. To draw CCGs, use the *ccg*[12] package. For dependency trees, there is the *tikz-dependency*[13] package.

5.11.3 Science Fiction

We already mentioned Ian Watson's *The Embedding* (1973), in which a not-very-ethical linguist gives drugs to brain-damaged children and makes them practice center-embedded sentences, such as "This is the malt that the rat that the cat that

[9] https://ctan.org/pkg/tikz-qtree
[10] https://ctan.org/pkg/qtree
[11] https://ctan.org/pkg/langsci-avm
[12] https://github.com/jasonbaldridge/cg-latex/blob/master/ccg.sty
[13] https://ctan.org/pkg/tikz-dependency

the dog worried killed ate." Here are some other examples of particular uses of syntax in Science Fiction narratives:

In Algis Budrys's *Silent Brother* (1956) [27, p. 297], the slow appropriation of the main human character's body and psyche by an alien is illustrated by one-word sentences: "*Blink* can't *think* blink rhythm I *think* blink trick think *blink* sink blink *wink*—CAN'T *THINK*!"

In Fredrick Brown's *The End* (1961) [27, pp. 299–300], after the 52 first words of the story, the main character pushes a button that reverses time so that the remaining story consists again of the same 52 words, read backward. Here is the central part of the story: "Pushing a button as he spoke, he said, "This should make time run backward run time make should this," said he, spoke he as button a pushing."

Finally, in Jerome Bixby's *Old Testament* (1964) [27, p. 298], after humans leave a planet, the natives write a mythology in which humans are "good gods." The language of the natives uses broken "petit-nègre" language, a colonialist reflex to show the cultural difference between natives and "visitors": "I happy. Everybody like little one. He friend of good gods. Other mothers take care of him. Let him drink."

5.12 Exercises

Exercise 4-1: Constituency parser comparison

In a corpus of 845 sentences `samples.txt`, find the structurally most different constituency trees produced by *stanza* and *spacy*, and explain their differences.

Exercise 4-2: How well do *stanza* and *spacy* parse Yoda?

We have chosen 40 sentences from the Yoda corpus[14] in file `yoda.txt`, and then have "de-yodified" them, in the sense that we have rewritten them as grammatical English senses, if possible without adding any additional words (file `adoy.txt`).

A good dependency parser should detect the role played by each word and its dependencies, even if the word order is not the standard SVO order of English. Evaluate the capacity of *spacy* and *stanza* to parse Yoda and draw a comparative plot. Explain some extreme cases.

Exercise 4-3: The Syntax of Lojban

Describe the syntax of the following text in the Lojban language:

> ni'o seldau lo selmi'ecatra fa lo du'u na ka'e katna vimcu lo stedu se cau lo nu da poi xadni zo'u ka'e katna vimcu lo stedu da

[14] https://www.kaggle.com/datasets/stefanocoretta/yoda-speech-corpus

which is the translation[15] by .xorxes. (Jorge Llambías) of a sentence of Lewis Carroll's *Alice's Adventures in Wonderland*:

> The executioner's argument was that you couldn't cut off a head unless there were a body to cut it off from.

adapted to our needs by Martin Bays. This exercise is a continuation of Exercise 3-4.

Exercise 4-4: Find Perfectly Ambiguous Sentences in English

At least two well-known French sentences can be qualified as "perfectly ambiguous." Every nongrammatical word in them can have two different POS tags, resulting in two interpretations with different meanings:

	Original					Translation
	La	belle	porte	le	voile	
Version 1		adjective	noun		verb	⌊The beautiful door hides it⌋
Version 2		noun	verb		noun	⌊The beautiful girl carries a veil⌋
	La	vieille	garde	le	lit	
Version 1		adjective	noun		verb	⌊The old guard reads him⌋
Version 2		noun	verb		noun	⌊The old woman stays in bed⌋

Find similar sentences in English.

Exercise 4-5: Find emoji that behave like noun phrases

Emoji can syntactically behave in various ways (see [24] and [59]). Most of them are sentence-final and are not part of the syntax tree of the sentence. But in some cases, they are used in place of noun phrases. We have extracted (and slightly adapted) 5,217 messages from the Kaggle *Emoji Sentiment* corpus[16] and stored them in file `comments-with-emoji.txt`. Find those that play the role of noun phrases in the syntax trees of sentences.

References

[1] Kazimierz Ajdukiewicz. "Die syntaktische Konnexität". In: *Studia Philosophica* 1 (1935), pp. 1–27.

[2] Yehoshua Bar-Hillel. "Some Linguistic Problems Connected With Machine Translation". In: *Philosophy of Science* 20 (1953), pp. 217–225.

[3] Katalin Bimbó. *Combinatory Logic, pure, applied and typed*. Discrete Mathematics and its Applications. Boca Raton, FL: CRC Press, 2012.

---

[15] https://alis.lojban.org

[16] https://www.kaggle.com/datasets/harriken/emoji-sentiment

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly, 2009.
[5] Guillaume Bonfante, Bruno Guillaume, and Guy Perrier. *Application of Graph Rewriting to Natural Language Processing*. Wiley, 2018.
[6] Robert D. Borsley and Kersti Börjars, eds. *Non-Transformational Syntax*. Wiley-Blackwell, 2011.
[7] Andrew Carnie. *Syntax. A Generative Introduction*. Ed. by 3. Wiley-Blackwell, 2013.
[8] Noam Chomsky. *Aspects of the Theory of Syntax*. Cambridge: MIT Press, 1965.
[9] Noam Chomsky. *Lectures on Government and Binding: The Pisa Lectures*. Dordrecht: Foris, 1981.
[10] Noam Chomsky. *Syntactic Structures*. The Hague: Mouton, 1957.
[11] Noam Chomsky. *The Minimalist Program*. MIT Press, 1995.
[12] Nelson Cowan. "What are the differences between long-term, short-term, and working memory?" In: *Prog. Brain. Res.* 169 (2008), pp. 323–338.
[13] William Croft. *Radical Construction Grammar: Syntactic Theory in Typological Perspective*. New York: Oxford University Press, 2001.
[14] David Crystal. *A Dictionary of Linguistics and Phonetics*. 6th ed. Blackwell Publishing, 2008.
[15] Anne De Roeck et al. "A Myth about Centre-Embedding". In: *Lingua* 58 (1982), pp. 327–340.
[16] Simon C. Dik. *Studies in Functional Grammar*. London Academic Press, 1980.
[17] Scott Drellishak. *The Embedding by Ian Watson*. `https://tenser.typepad.com/tenser_said_the_tensor/2006/04/the_embedding_b.html`. 2006.
[18] Julia Ecklar. "The Kobayashi Maru". In: *Star Trek Pocket Books* 47 (1989).
[19] Michel Fayol, Pierre Largy, and Patrick Lemaire. "When cognitive overload enhances subject-verb agreement errors. A study in French written language". In: *The Quarterly Journal of Experimental Psychology A* 47 (2007), pp. 437–464.
[20] Charles J. Fillmore. "The Mechanisms of "Construction Grammar"". In: *Proceedings of the Fourteenth Annual Meeting of the Berkeley Linguistics Society*. 1988, pp. 35–55.
[21] Lyn Frazier and Janet Dean Fodor. "The sausage machine: A new two-stage parsing model". In: *Cognition* 6 (1978), pp. 291–325.
[22] Gerald Gazdar et al. *Generalized Phrase Structure Grammar*. Oxford: Blackwell, 1985.
[23] Giorgio Graffi. *200 Years of Syntax*. Amsterdam: John Benjamin, 2001.
[24] P.G. Grosz, G. Greenberg, C. De Leon, et al. "A semantics of face emoji in discourse". In: *Linguist. and Philos.* (2023). `doi:10.1007/s10988-022-09369-8`.
[25] Michael Halliday. *An Introduction to Functional Grammar*. Hodder Arnold, 1985.
[26] Yannis Haralambous. "Ne vous moquez pas de l'oiseau moqueur : un aperçu de la logique combinatoire, avec des applications en linguistique mathématique". In: *Quadrature* 113 (2019), pp. 22–34.
[27] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. "Graphemic and Graphetic Methods in Speculative Fiction". In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.
[28] David G. Hays. "Dependency Theory: A Formalism and Some Observations". In: *Language* 40 (1964), pp. 511–525.
[29] David G. Hays. "Grouping and Dependency Theories". In: *Proceedings of the National Symposium on Machine Translation, UCLA February 1960*. 1960, pp. 258–266.
[30] David G. Hays. *Grouping and dependency theories*. P-1910. RAND Corporation, 1960.
[31] David G. Hays. *Introduction to Computational Linguistics*. Vol. 2. Mathematical Linguistics and Automatic Language Processing. Elsevier, 1967.
[32] Kees Hengeveld and J. Lachlan Mackenzie. *Functional Discourse Grammar: A Typologically-Based Theory of Language Structure*. Oxford: Oxford University Press, 2008.
[33] Roger J. Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators. An Introduction*. Cambridge University Press, 2008.

[34] Paul Hopper. “Emergent Grammar and Temporality in Interactional Linguistics”. In: *Constructions*. Ed. by P. Auer and S. Pfänder. Berlin: De Gruyter, 2011, pp. 22–44.
[35] Ray Jackendoff. *Introduction to the X-bar Convention*. Indiana University Linguistics Club, 1974.
[36] Yan Jianwei and Haitao Liu. “Which annotation scheme is more expedient to measure syntactic difficulty and cognitive demand?” In: *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*. 2019.
[37] Arvind K. Joshi, Leon S. Levy, and Masako Takahashi. “Tree Adjunct Grammars”. In: *Journal of Compuer and System Science* 10 (1975), pp. 136–163.
[38] Ronald Kaplan and Joan Bresnan. “Lexical-Functional Grammar: A Formal System for Grammatical Representation”. In: *The Mental Representation of Grammatical Relations*. Ed. by J. Bresnan. The MIT Press, 1982, pp. 173–281.
[39] Martin Kay. “Functional grammar”. In: *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*. 1979, pp. 142–158.
[40] Koenraad Kuiper and Jacqui Nokes. *Theories of Syntax. Concepts and Case Studies*. Bloomsbury, 2014.
[41] Adrienne LaFrance. “An Unusual Way of Speaking, Yoda Has. Hmmm?” The Atlantic (Boston). December 18, 2015. https://perma.cc/HAW4-VYBY.
[42] George Lakoff. “Linguistic gestalts”. In: *Papers from the Thirteenth Regional Meeting, Chicago Linguistic Society, April 14–16, 1977*. Ed. by Beach W.A., Fox S.E., and Philosoph S. Chicago, Illinois: Chicago Linguistic Society, University of Chicago, 1977, pp. 236–287.
[43] Joachim Lambek. “The mathematics of sentence structure”. In: *Amer. Math. Monthly* 65 (1958), pp. 154–170.
[44] Ronald W. Langacker. *Foundations of Cognitive Grammar I. Theoretical Prerequisites*. Stanford: Stanford University Press, 1987.
[45] Géraldine Legendre, Jane Grimshaw, and Sten Vikner. *Optimality-theoretic syntax*. MIT Press, 2001.
[46] Géraldine Legendre, Yoshiro Miyata, and Paul Smolensky. “Can connectionism contribute to syntax? Harmonic Grammar, with an application”. In: *Proceedings of the 26th regional meeting of the Chicago Linguistic Society*. Ed. by M. Ziolkowski, M. Noske, and K. Deaton. Chicago: Chicago Linguistic Society, 1990, pp. 237–252.
[47] Jacqueline Léon. *Automating Linguistics*. Springer, 2021.
[48] Marie-Catherine de Marneffe et al. “Universal Dependencies”. In: *Computational Linguistics* 47 (2021), pp. 255–308.
[49] André Martinet. *A Functional View of Language*. Clarendon Press, 1962.
[50] Tim Osborne and K. Gerdes. “The status of function words in dependency grammar: A critique of Universal Dependencies (UD)”. In: *Glossa* 4 (2019).
[51] Timothy Osborne. *A Dependency Grammar of English. An introduction and beyond*. John Benjamins Publishing Company, 2019.
[52] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Stanford: CSLI, 1994.
[53] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics*. Vol. 1. Stanford: CSLI, 1987.
[54] Geoffrey K. Pullum. “Word order universals and grammatical relations”. In: *Grammatical Relations*. Ed. by P. Cole and J. M. Sadock. Vol. 8. Syntax and Semantics. 1977, pp. 249–277.
[55] Peng Qi et al. “Stanza: A Python Natural Language Processing Toolkit for Many Human Languages”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020.
[56] Ivan A. Sag, Thomas Wasow, and Emily M. Bender. *Syntactic Theory. A Formal Introduction*. Ed. by 2. Stanford: CLSI, 2003.
[57] Raymond Smullyan. *To mock a mockingbird, and other logic puzzles including an amazing adventure in combinatory logic*. New York: Knopf, 1985.

[58] Herbert Weir Smyth. *A Greek Grammar for Colleges*. American Book Company, 1920.

[59] Chenchen Song. "Sentence-final particle vs. sentence-final emoji: The syntax-pragmatics interface in the era of CMC". In: *Proceedings of Grapholinguistics in the 21st Century, 2022*. Ed. by Yannis Haralambous. Grapholinguistics and Its Applications. to appear. Brest: Fluxus Editions, 2023.

[60] Richard Sproat. *A Computational Theory of Writing Systems*. Studies in Natural Language Processing. Cambridge: Cambridge University Press, 2006.

[61] Mark Steedman. "Combinatory grammars and parasitic gaps". In: *Natural Language and Linguistic Theory* 5 (1987), pp. 403–439.

[62] Mark Steedman. *The Syntactic Process*. The MIT Press, 2000.

[63] Shira Tal and Inbal Arnon. "Redundancy can benefit learning: Evidence from word order and case marking". In: *Cognition* (2022).

[64] Lucien Tesnière. *Éléments de syntaxe structurale*. Éditions Klincksieck, 1959.

[65] Lucien Tesnière. *Elements of Structural Syntax*. Translated by Timothy Osborne and Sylvain Kahane. John Benjamins, 2015.

[66] Robert D. Van Valin Jr. and William A. Foley. "Role and Reference Grammar". In: *Current approaches to syntax*. Ed. by E. A. Moravcsik and J. R. Wirth. Vol. 13. Syntax and semantics. New York: Academic Press, 1980, pp. 329–352.

[67] Paul Warren. *Introducing Psycholinguistics*. Cambridge University Press, 2013.

[68] William Woods. "Transition Network Grammars for Natural Language Analysis". In: *Communications of the ACM* 13 (1970), pp. 591–606.

Chapter 6
Semantics (and Pragmatics)

Semantics is the branch of linguistics studying *meaning*.

In this chapter, we will see the various ways meaning can be analyzed and represented, be it on the level of elementary units such as morphemes and words (*lexical semantics*) or on the level of sentences/utterances (*phrasal semantics*) and discourse (*discourse semantics*).

Pragmatics (the "wastebasket of the study of meaning," according to [47, p. 474]) is the study of utterances in context. We have added it to the chapter's title between parentheses to challenge the distinction between semantics and pragmatics, a distinction that has historical origins rather than being an epistemological imperative [66].

It is important to notice that semantics deals with meaning only in the context of *language*. Suppose we enlarge our scope to meaning carried by other artifacts or concepts or activities, such as (in alphabetical order) brand logos, clothes, computing, diagrams, flags, food, games, gardens, history, human bodies, identity, ideology, maps, music, politics, virtual worlds, or whatever is not language. In that case, we leave the domain of semantics and enter another utterly exciting discipline called *semiotics*[1]. Semiotics is outside the scope of this book; see the Further Reading section for some valuable links.

6.1 Sense Relations

Before we enter into specific semantic theories, let us give some types of relations between lexemes or terms, the common point of which is that they all have an "-onymy" suffix (from the Greek "ὄνομα"$_{\text{EL}}$ ⌊name⌋), like "hyponymy," "homonymy," etc. There is some debate about whether these relations are linguistic or encyclopedic. Let us start with the major sense relations:

[1] The terms "semantics" and "semiotics" both are based on the Greek stem "σημ-"$_{\text{EL}}$—the former term is akin to the verb "σημαίνω"$_{\text{EL}}$ ⌊to signify⌋ and the latter to the noun "σημεῖον"$_{\text{EL}}$ ⌊sign⌋.

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_6

hyponymy: when we have a "kind of" relation (e.g., "apple" and "banana" are *hyponyms* of "fruit," in the sense that all apples and bananas are fruits but not all fruits are apples and bananas);

hyperonymy: the inverse of hyponymy (e.g., "vehicle" is a *hyperonym* of "bus": all busses are vehicles—not all vehicles are busses);

synonymy: a hyponymy that is also a hyperonymy, and vice-versa (e.g., "car" and "automobile," since all cars are automobiles and all automobiles are cars). This is an *extensional definition* of synonymy since it is based on the sets of instances of the two concepts. Synonymous lexems can very well have different connotations or can belong to different registers ("automobile" is more formal than "car"). Another definition of synonymy is by *interchangeability*: Lyons [54, p. 448] defines *total synonymy* as the relation between two lexemes that are interchangeable in all contexts, otherwise we have *partial synonymy*;

antonymy: having the opposite meaning, in some sense (e.g., "good" and "evil" are the founding antonyms of many religions). Notice that depending on the context, antonyms can vary: the antonym of "red" is "green" for traffic lights, "white" for wine, "black" in accounting (as in "Black Friday");

homonymy: the same written and spoken form has several unrelated meanings (e.g., "bank" as a financial institution, as part of a river, as a bench or as a row of items; similarly, in French, "voler"$_{\mathrm{FR}}$ as *to steal* or *to fly*; similarly in German, "Gericht"$_{\mathrm{DE}}$ as a court or as a meal);

meronymy: the being-part-of relation (e.g., "finger" is a meronym of "hand"). Notice that contrarily to formal languages such as mathematics where the inclusion relation $\subset$ is transitive, meronymy is not transitive in natural language: if (Chomsky's) finger is part of Chomsky and Chomsky is part of MIT, Chomsky's finger is *not* a meronym of MIT, since the members of MIT are people and not fingers. Priss [68] distinguishes four types of meronymy:

- the relation of a functional component to a whole ("lung" $\to$ "body");
- the relation of a segment to a (pre-existing) whole ("page" $\to$ "book");
- the relation of membership to a set ("person" $\to$ "team");
- the relation of a subset to a set ("children" $\to$ "audience"). Notice that even though a "child spectator" is a special kind of "spectator" (and hence, a hyponym) and an "audience of children only" a special kind of "audience" (again a hyponym), the group of "children in the audience" is a meronym of "audience";

holonymy: the inverse of *meronymy*.

Here are some more exotic sense relations:

paronymy: lexemes looking alike but with unrelated meanings (e.g., "continuous" vs. "contiguous," "upmost" vs. "utmost," "alternately" vs. "alternatively," "intension" vs. "intention," etc.). Notice that "farther" and "further" are not paronyms since they are strongly related;

contronymy or *autoantonymy*: when a word has two contradictory senses, such as "apology," "consult," "bill," etc.

retronymy: when a new term is introduced to replace an older term that has become ambiguous, e.g., “acoustic guitar” is a retronym of “guitar” (because before the invention of electric guitars, all guitars were acoustic);

collocation: Lyons [55, p. 125] considers collocation as a sense relation. He gives the example of “rancid” being applied only to “butter” and “addled” only to “egg”.

In the following, we will discuss four different approaches to semantics, all of which result in applications in Natural Language Processing. The classification of these approaches, inspired by Geeraerts [25], is as follows: *structuralist* approaches, *cognitive* approaches, a middle ground between them that he calls *neostructuralist*, and finally, *truth-conditional* approaches. This classification is far from perfect, as there are interactions between the families. We will mention them whenever possible.

6.2 Structuralist Approaches to Semantics

6.2.1 Lexical Field Theory

In 1934, the two German linguists Trier and Porzig independently defined *fields*, i.e., sets of lexemes with complementary definitions.

On the one hand, Trier [81] defines *lexical fields* as sets of words in which “words acquire their meaning through their relationship to other words in the same field,” and by relationship, he means contrast and inclusion. What is interesting is that Trier defines these fields as systems in which lexemes share their “meaning content” so that when the meaning of one lexeme is extended, those of the others are narrowed [46, p. 68]. He applies this notion to the historical evolution of specific word groups to explain the phenomena of simultaneous meaning shifts. Kronenfeld & Rundblad [46] give an example of lexical field:

{“river,” “stream,” “brook,” “beck” (UK), “creek” (US)}.

This field has only five entries, while in Old English, it contained over 100 lexemes that differed by origin and specialized meaning.

Porzig [67], on the other hand, defines *semantic fields* as being sets of collocated words such as

{“lick,” “tongue”},

i.e., words that are statistically frequently used in the same phrase.

In 1956, the French linguist Giraud [32] expanded Trier’s field membership criteria to morphological criteria, such as derivation and phonological criteria, such as homophony. He calls the new fields, *morphosemantic fields*.

One of the weaknesses of field theories, specifically Trier’s theory, is that a language is presented as a “mosaic of meanings,” a metaphor that implies the absence of empty slots, “holes,” in the mosaic. In 1974, Lehrer [51] drew the following table to show the presence of empty slots based on four features:

	conducted warmth (oven)	radiated warmth (fire)	hot surface (pan)
+ water, – oil, – vapor	boil	boil	boil
+ water, – oil, + vapor	steam	steam	steam
– water, + oil	(oven-fry)	∅	fry
– water, – oil	bake	broil	∅

As she remarks, there is no word for preparing food in a pan without water and oil or for cooking with oil on a flame. See the appendix of Kronenfeld & Rundblad [46] for a study of the “river” lexical field.

6.2.2 Componential Analysis, Formal Concept Analysis

The table of the previous section is an example of the *componential analysis* process, which consists of finding a small set of features that describe and discriminate the lexemes of a given set. This approach that emerged both in the US and in Europe in the fifties and early sixties is heavily inspired by phonology, where a small number of features allows the description of the entire phonological system of a language. This approach is only useful when the number of features is significantly smaller than ☞ 15 the number of emes to describe (otherwise, one can always use as many features as emes and define each eme X as the unique eme having feature $+X$).

Here is an example of the componential analysis of six verbs related to the exchange of items between people:

English	French	German	Greek	exchange	to give	to take	for money	permanent
to buy	acheter	kaufen	ἀγοράζω	+	–	+	+	+
to sell	vendre	verkaufen	πουλάω	+	+	–	+	+
to lend	prêter	ausleihen	δανείζω	+	+	–	–	–
to borrow	emprunter	leihen	δανείζομαι	+	–	+	–	–
to lease	louer	vermieten	νοικάζω	+	+	–	+	–
to rent	louer	mieten	νοικιάζω	+	–	+	+	–

Notice that, in French and Greek, the same lexeme is used for leasing and renting.

Componential analysis has been formalized (and extensively used in knowledge representation) by the German mathematicians Ganter and Wille in 1982 [23], under the name *Formal Concept Analysis* (FCA). Using the mathematical theory of lattices, Ganther and Wille redefine the notions of *context*, *concept*, *extension*, and *intension* in a purely formal way. The terms “extension” and “intension” require some explanation: the *extension* of a concept is the set of all of its instances, e.g., if we take the concept CAT, its extension is the set of all cats (past, present or future, real or imaginary), and defining the concept of CAT in an extensional manner is to provide a set with representatives (names, IDs, DNA signatures, etc.) of all cats. The 's' in the word "intension" is intentional (pun!) since "intension" is meant to be an antonym (or at least, a complement) of "extension": an *intension* is the description of a concept *by its properties*. An example in mathematics best illustrates the two notions: the set of even positive integers less or equal to 10 has the *extension* $\{0, 2, 4, 6, 8, 10\}$ and the *intension* $\{n \in \mathbb{N}_0 \mid \exists p \in \mathbb{N}_0, n = 2p, n \leq 10\}$.

The starting point of FCA is the notion of *context*, defined as a triplet of sets (G, M, I) where the members of G are objects (from the German "Gegenstand"$_{\mathrm{DE}}$), the members of M are features (from the German "Merkmal"$_{\mathrm{DE}}$), and I is a set of binary relations $I \subseteq G \times M$. Having $(g, m) \in I$ means that "object g has feature m".

Using solely the notion of context, FCA defines *concepts*, *extensions*, *intensions* and builds a mathematical structure with very nice properties: a *Galois lattice*.

To illustrate the definitions that follow, we will use an example taken from Ganter *et al.* [23, p. 27]:

	lays eggs	is nightly	is dangerous	is domestic	is aquatic	is feline
alligator	+	–	+	–	+	–
cat	–	+	–	+	–	+
shark	–	–	+	–	+	–
tarantula	+	+	+	–	–	–
lion	–	–	+	–	–	+

We have chosen features and objects in a way that features discriminate all objects, and objects discriminate all features (i.e., there are neither two identical lines nor two identical columns in the table). In our example, the objects are $G = \{$aligator, cat, shark, tarantula, lion$\}$ and the features are $M = \{$eggs, nightly, dangerous, domestic, aquatic, feline$\}$.

For subsets G_1 and M_1 of G and M we define an operator $\circ^I$ (an exponent "I") called *derivation*, in the following way:

$$G_1^I = \{m \in M \mid \forall g \in G_1, (g, m) \in I\}$$
$$M_1^I = \{g \in G \mid \forall m \in M_1, (g, m) \in I\}.$$

This means that G_1^I is the set of features shared by *all* elements of G_1 (if any) and M_1^I is the set of objects having *all* features of M_1 (if any). Notice that G_1^I is a subset of M and M_1^I a subset of G. In other words, the derivation operator sends subsets of G to subsets of M and vice-versa.

In our example, if we take $G_1 = \{\text{shark}, \text{lion}\}$, then $G_1^I = \{\text{dangerous}\}$ since it is the only common feature (among those of M) of a shark and a lion, and if $M_1 = \{\text{dangerous}, \text{aquatic}\}$, then $M_1^I = \{\text{aligator}, \text{shark}\}$ since these are the only animals (in G) that are both dangerous and aquatic. We will now use the derivation operator to define the notions of *concept*, *extension*, and *intension*:

A *concept* (X, N) is a pair of sets such that $X \subseteq G$, $N \subseteq M$ with the following symmetry property: $X^I = N$ and $N^I = X$. We call X the *extension* and N the

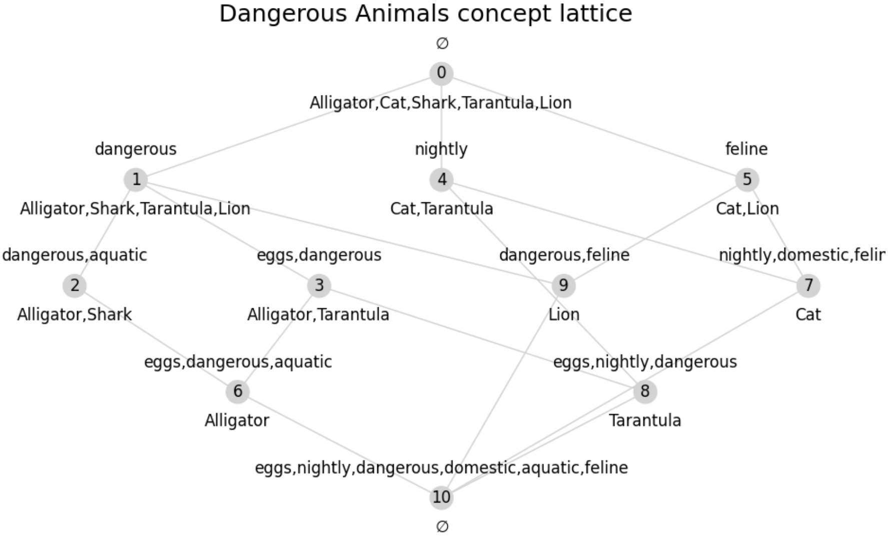


Fig. 6.1 The lattice of the example of dangerous animals, calculated and drawn by the Python package *FCApy*. Above the vertices, we display features and, beneath them, objects.

intension of the concept. In other words, if we take a set of features and call it the intension of the concept, then its extension will be the set of all objects with these features in common and only these features in common.

In our example, we can take $(X, N) = (\{\text{cat}, \text{lion}\}, \{\text{feline}\})$. Indeed, "cat" and "lion" are the only animals having the "feline" feature, and this feature is the only feature they have in common. We, therefore, have a concept, the intension of which is "having the feline property" and the extension of which is the set containing "cat" and "lion."

By the definition of concept, we have $X^{II} = N^I = X$ and $N^{II} = X^I = N$, i.e., a concept is a set of objects X and a set of features N, on which $I \circ I$ is idempotent.

In the set of concepts of a given context, we have a partial order $\leq$ defined as follows:

$$(X_1, N_1) \leq (X_2, N_2) \iff X_1 \subseteq X_2 \quad (\iff N_1 \supseteq N_2).$$

Intuitively, this means that a concept is "larger" when it contains more instances, but for that, we need to request fewer features (features act like restrictions). Two extreme cases can happen: the one where we request no feature and get the entire context, and the other where we request all features and probably get no object. In our example, there is no such object, or it could be a very special object that happens to have all features, such as a monster imagined by travelers in Antiquity [15], that is feline but also lays eggs, lives at night, and is both domestic and dangerous…).

In Fig. 6.1, the reader can see the lattice of our example. On this lattice, one can explore the structure of the context by following the different paths from top to bottom or vice-versa and observe what happens when features are added or removed.

This kind of lattice is called a *Galois lattice* after the great French mathematician Évariste Galois (1811–1832). Galois was working on groups, but his theory was applied to partially ordered sets in 1944 by the Norwegian mathematician Ore [65]. He defines sets of objects G_1 or features M_1 to be *closed* when $I^{-1}(I(G_1)) = G_1$ and $I(I^{-1}(M_1)) = M_1$, and he defines G_1 and M_1 to be *Galois-connected* if $I(G_1) = M_1$ and $I^{-1}(M_1) = G_1$. In our case, closure can be easily shown, and the fact of G_1 and M_1 being Galois-connected is when we call (G_1, M_1) a concept.

Priss [69] presents an extended version of FCA called *relational concept analysis* that adds more relations between concepts besides the hierarchical relation induced by the partial order of the lattice.

6.2.3 Relational Semantics

In [54, p. 443], Lyons states that

> the sense of a lexical item may be defined to be not only dependent on but identical to the set of relations that hold between the item in question and the other items in the same lexical system.

The argument he gives to support this claim is one of relativity in the physical world: when we ask, "What is the length of this object?" in reality, we ask a question of comparison: "How does this object compare with the platinum-iridium standard bar, kept in Paris?". He concludes:

> The question "What is the sense of x?" [...] is methodologically reducible to a set of questions, each of which is relational: "Does sense-relation R_i hold between x and y?"

In his books, rather than developing a semantic theory based solely on sense relations, Lyons advocates (and criticizes) componential analysis. But relying on semantic relations to describe the meaning of lexemes is premonitory of knowledge ☞ 311
graphs as used today.

6.2.4 WordNet

WordNet is a semantic network with a graph structure. In this structure, vertices are ☞ 207
synsets and edges, relations such as hyperonymy/hyponymy, synonymy, meronymy/holonymy, and antonymy. Before we describe the notion of a synset, here is some history: the first version of WordNet resulted from a project at Princeton University funded by the US Office of Naval Research. In 2006, in version 3.1, the original WordNet project stalled. In the meantime, in 2000, the Global WordNet Association (GWA), a noncommercial organization, was founded to foster the development of wordnets worldwide. Nowadays, on the site[2] of the GWA, one can find individual

[2] http://globalwordnet.org/about-gwa/

WordNets as parts of an Open Multilingual Wordnet (OMW) version 1, and also the description of a grid of wordnets (for 28 languages), called OMW version 2 [95].

Let us now consider the basic element of WordNet, the *synset*. It is a set of numbered lexemes, called *lemmas* in WordNet, such that the meaning of the synset is the common meaning of the lemmas. Here is an example of four synsets, followed by *glosses* (definitions and optional examples, which we don't give here):

— car_1: car auto automobile machine motorcar | a motor vehicle with four wheels; usually propelled by an internal combustion engine
— car_2: car railcar railway_car railroad | a wheeled vehicle adapted to the rails of railroad
— car_3: car gondola | the compartment that is suspended from an airship and that carries personnel and the cargo and the power plant
— car_4: car elevator_car | where passengers ride up and down; "the car was on the top floor"

The "car" example is nice because every synset has different lemmas. Things don't always work out that way. Let us take the example of "machine":

— machine_1: machine | any mechanical or electrical device that transmits or modifies energy to perform or assist in the performance of human tasks
— machine_2: machine | an efficient person
— machine_3: machine | an intricate organization that accomplishes its goals efficiently
— machine_4: machine simple_machine | a device for overcoming resistance at one point by applying force at some other point
— machine_5: machine political_machine | a group that controls the activities of a political party
— machine_6: car auto automobile machine motorcar | a motor vehicle with four wheels; usually propelled by an internal combustion engine

As we see, "machine_6" is the same synset as "car_1." As for synsets 1, 2, and 3 of "machine," they are not distinguished by their lemmas but by their glosses.

This seems complex, but this complexity is inherent to English.

Synsets are the nodes of the graph. As for relations, we have a lot of hyperonymy relations (and hence also of hyponymy relations by taking the same edges in the opposite direction). The particularity of WordNet is that it has a single root for nouns, called "entity_1." Hyperonym paths normally end at that synset, the most abstract and general notion possible. Here are the hyperonym paths of the four "car_i" synsets:

— car_1: motor_vehicle_1 $\text{self-propelled\_vehicle}_1$ wheeled_vehicle_1 container_1 vehicle_1 instrumentality_3 conveyance_3 artifact_1 whole_2 object_1 physical_entity_1 entity_1
— car_2: wheeled_vehicle_1, etc.
— car_3: compartment_2 room_1 area_5 structure_1 artifact_1, etc.
— car_4: compartment_2, etc.

Hyperonym paths allow us to compare the synsets. On the one hand, "car_3" and "car_4" have exactly the same hyperonyms. They are siblings, hyponyms of "compartment_2". On the other hand, "car_1" and "car_3" are quite different. The former, "car_1" is a kind of motor vehicle, which is a kind of self-propelled vehicle, which is a kind of wheeled vehicle, which is a kind of container, which is a kind of vehicle, which is a kind of instrumentality, which is a kind of conveyance, which is a kind of artifact. The latter, "car_3" is a compartment, which is a kind of room, which is a kind of area, which is a kind of structure, which is a kind of artifact. Both are artifacts, but to get there, we had to move up eight steps for the former and five for the latter.

It becomes evident that the length of the path that connects two nodes is inversely proportional to their similarity in the sense of the hyperonmy/hyponym relation. This is called an "ontological similarity." It never really became a semantic relatedness measure because WordNet can be very densely populated in some areas and much less in others, and this, of course, affects the length of the path and the quality of the result.

WordNet's benefits are that it has been built manually, under the supervision of linguists, and that synsets are sufficiently neutral to be adapted to all tasks. Its main drawback is that it starts being useful once you know what synset corresponds to each word in your corpus. Disambiguating words to obtain the appropriate synsets is not a trivial task.

Let us see now how WordNet is implemented in Python. It is part of the Python *nltk* package. Here is how to obtain the data presented above:

```
1 from nltk.corpus import wordnet as wn
2
3 print(wn.synsets('car'))
4 hyp = lambda s:s.hypernyms()
5 for digit in ["01","02","03","04"]:
6     print(wn.synset('car.n.'+digit).lemmas())
7     print(wn.synset('car.n.'+digit).definition())
8     print(list(wn.synset('car.n.'+digit).closure(hyp)))
```

Some explanations: on line 3 we ask for all synsets with a lemma "car," or containing the string "car." The results we get are

```
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'),\
Synset('car.n.04'), Synset('cable_car.n.01')]
```

In `car.n.01`, the `n` stands for "noun," and the number is the number of the lemma. Besides "car$_i$" for $1 \leq i \leq 4$, we also get "cable_car$_1$," which contains "car." On line 6, we request the four synsets "car$_i$." On line 7, we request the definition of each synset. The definition is only part of the gloss. In many cases there are also examples obtained by `examples()`. Finally, on line 4, we define a function called `hyp`, that sends each synset to its hyperonyms. We use it on line 8, where we request the *closure* of a synset for the `hyp` relation. Here, closure means that we take all hyperonyms until none is left. Usually, this happens when we arrive at the "entity$_1$" synset. This has produced the chains of hyperonyms we saw on p. 144.

See §8.2.4 for an approach to WordNet using graph theory packages such as *networkx* and *graph-tool*.

6.3 Neostructuralist Approaches to Semantics

6.3.1 Wierzbicka's Natural Semantic Metalanguage

To describe the meaning of a lexeme, we can use componential analysis or natural language sentences. Using this scheme, circularity is unavoidable. To avoid circular-

Table 6.1 English representation of the universal Natural Semantic Metalanguage Semantic Primes (as of 2014 [30, p. 12]). Words separated by slashes are allolexes of the same concept.

Substantives:	I, YOU, SOMEONE, PEOPLE, SOMETHING/THING, BODY
Relational substantives:	KIND, PART
Determiners:	THIS, THE SAME, OTHER/ELSE
Quantifiers:	ONE, TWO, SOME, ALL, MUCH/MANY, LITTLE/FEW
Evaluators:	GOOD, BAD
Descriptors:	BIG, SMALL
Mental predicates:	THINK, KNOW, WANT, FEEL, SEE, HEAR
Speech:	SAY, WORDS, TRUE
Actions, events, movement:	DO, HAPPEN, MOVE
Location, existence, specification:	BE (SOMEWHERE), THERE IS, BE (SOMEONE/SOMETHING)
Possession	(IS) MINE
Life and death:	LIVE, DIE
Time:	WHEN/TIME, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT
Space:	WHERE/PLACE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE, TOUCH
Logical concepts:	NOT, MAYBE, CAN, BECAUSE, IF
Intensifier, augmentor:	VERY, MORE
Similarity:	LIKE/AS/WAY

ity, we can adopt a rule, such as “the lexemes used in the description shall be *simpler* than the lexeme described.” This raises the question: if we do this and always choose simpler lexemes, will we ever reach a minimum level of complexity?

Wierzbicka [99, p. 72] had the idea of finding the lexemes or morphemes that occur at the *lowest* level of complexity. She calls them *semantic primes*. She set herself the task of finding the common “simple” lexemes/morphemes in all languages. And by “all,” she really means *all* languages and not just the “usual suspects,” having millions of speakers.

In [99, p. 35] Wierzbicka gives the example of the word “brother.” This word may seem very basic to us, but there are actually languages in which it does not exist, e.g., in Turkish, the word “kardeş”$_{\text{TK}}$ means both “brother” and “sister,” and there is no more specific word. Necessarily these languages have other means to signify the concept of “brother,” and therefore it is not a semantic prime. In Table 6.1, we display the list of semantic primes that Wierzbicka and her colleagues have gathered after 30 years of investigation (her first publication on the subject is from 1972, and the list of Table 6.1 is its 2014 version, containing 64 lexemes).

Besides the common lexical core, Wierzbicka [98] states that languages have a common grammatical core so that

> at the heart of all languages there lies a mini-language, with as many realizations as there are languages. Thus, there is a mini-English that can be called “NSM English,” a matching mini-Russian (“NSM Russian”), and so on. Each of these mini-languages can serve as a culture-free metalanguage for analyzing meanings and ideas, for comparing languages and cultures, and for elucidating ideas in any domain of social science, [98, p. 33]

where NSM stands for *Natural Semantic Metalanguage*. These mini-languages contain the 56 universal semantic primes, more complex lexemes (called *semantic molecules*) built out of the semantic primes, and grammatical rules to combine them and produce meaning.

It should be noted that sometimes there are more than one lexemes needed for a given semantic prime, such as "me" and "I" for the semantic prime I, or "أَنتَ"$_{\text{AR}}$ /ʔanta/ and "أَنتِ"$_{\text{AR}}$ /ʔanti/ for YOU (masculine vs. feminine). These are called *allolexes* (similarly to "allophones," "allographs" and "allomorphs"). Also, interestingly, it is not assumed that the lexical representations of semantic primes must be distinct in all languages: as a counterexample, in the Austronesian language Mangap-Mbula of Papua New Guinea (a language having inclusive and exclusive duals and plurals), the same morpheme "-so" is used both for WANT and for SAY [29, p. 32].

Once the ingredients of a mini-language have been established, other words can be defined by an operation called "reductive paraphrase," using either semantic primes or molecules and grammar rules. Here is, for example, the definition of "amazement" [99, p. 179], where we use small caps for primes from Table 6.1:

X FEELS SOMETHING
sometimes a person THINKS SOMETHING like THIS:
    SOMETHING is HAPPENing NOW
    I DIDN'T KNOW BEFORE NOW: THIS CAN HAPPEN
    I WANT to KNOW MORE about it
BECAUSE of THIS, this person FEELS SOMETHING
X FEELS SOMETHING like THIS

(the graphetic method of indentation is used to delimit logical blocks).

As Geeraerts [25, p. 137] puts it, "[t]he Natural Semantic Metalanguage approach to lexical analysis smacks of idealism." Nevertheless, there have been applications of this approach to Artificial Intelligence (see, e.g., Fahndrich [16]).

6.3.2 Conceptual Semantics

In [39, p. 336], Jackendoff criticizes NSM and other decompositional trends by drawing a parallel with physics (referring to atoms, particles, quanta and the like):

> An important question often raised about lexical decomposition is the justification of primitives. Where does decomposition end? When do we know to stop? I don't think the parallel question in physics worries its practitioners too much. The history of the last two hundred years is a continual quest to explain deeper and deeper regularities. Semantics is just getting started; let's show some patience.

Pursuing this analogy even further, he concludes that if primitives exist they can hardly be real words but rather "technical primitives," artifacts by which we can analyze word meaning.

For example, let us take the two sentences [39, pp. 364–366]:

John went into the room.

John entered the room.

Jackendoff uses primitive types such as Object, Place, Path, and Event to formalize them. He considers that the verb "to go" takes two arguments: the subject that is of type Object, and information about either where the subject is going or from where and towards where the subject is going, in short: an argument of type Path. In type theory notation, the verb "to go" is a function with the following signature:

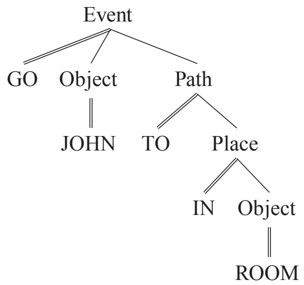
GO: <(Object,Path),Event>,

where inside the angular brackets, we have the input and output types; the input type is a pair of types for the two verb arguments.

The structure we obtain is

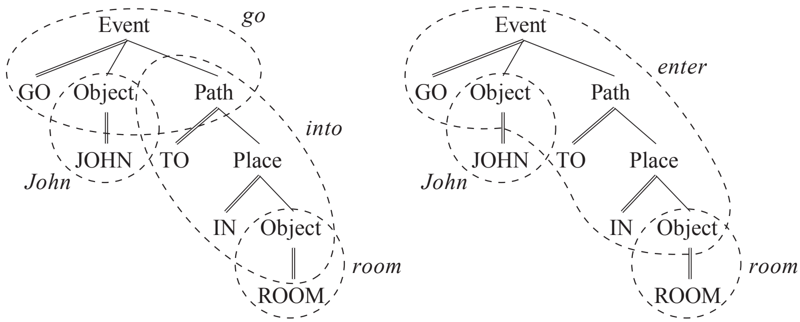
$[_{\text{Event}}\text{go }([_{\text{Object}}\text{john}]_{\text{i}}, [_{\text{Path}}\text{to }([_{\text{Place}}\text{in }([_{\text{Object}}\text{room}]_{\text{m}})]_{\text{k}})]_{\text{j}},$

where indices "Event," "Object," "Path," and "Place" of opening brackets are types. Indices of closing brackets denote referents (we have four referents: i for John, m for the room, k for the path "into the room," and j for the entire event). This structure (without explicit types and referent indices) is represented as follows:

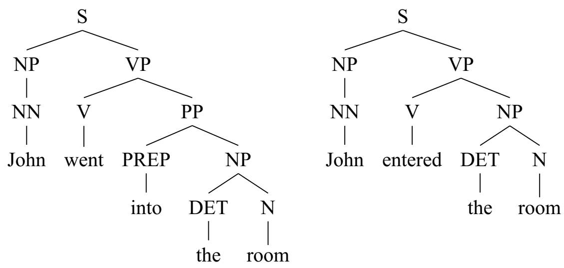


(double lines connect functions with their types, and simple lines stand for their arguments). The branches of this diagram are called *conceptual constituents*.

This diagram is equally valid for the "John entered the room" sentence, which is synonymous to "John went into the room". Here are the contributions of the words of the two sentences to the same semantic structure diagram:



The sentences have the same semantic tree, but their syntax trees differ:



As the reader may have noticed, both verbs ("to go" and "to enter") are represented by the same function, "to go." This is because Jackendoff considers that "to go" is a core function of the conceptual organization of events—as for "to enter," he considers that this verb "incorporates" the functions of Path and of Place. The semantic trees remain unchanged since the functions are the same. They are only distributed differently on the lexical and syntactic levels.

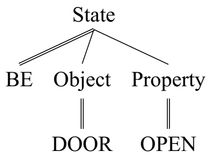
In the examples above, we display all functions corresponding to lexemes. Here is another example [73, p. 267]:

The door was open.

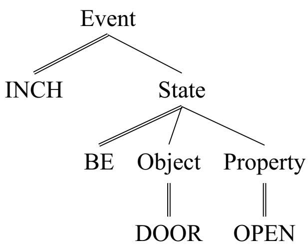
The door opened.

John opened the door.

The first sentence requires the function "to be," which has the signature <(X,Y), State>, meaning that it takes two arguments of arbitrary type and returns the type State. "Open" is of type Property, so that we get the diagram



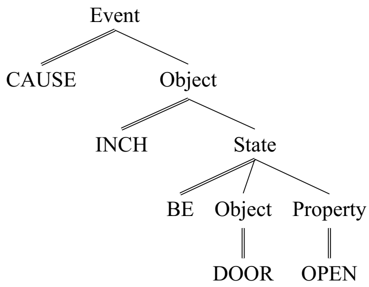
In the sentence "The door opened," we must show that the state of being open has just begun. The function for doing that is called *inchoative* (INCH). Beginning a state is an event (a state has duration, while the beginning of the state is instantaneous, or at least significantly shorter) so that the signature of INCH is <State, Event>. The diagram we obtain is as follows:



As the reader has undoubtedly noticed, this is the same as the previous, but with an additional Event of type INCH at its root. It could be interpreted as "the door began to be in an open state," and this is, in fact, all we can say since we don't know whether the door opened by itself or if somebody opened it. And this leads us to the third sentence, "John opened the door," where John caused the door to enter the open state. For that, we will use the function CAUSE with the following signature:

CAUSE: <(Object/Event,Event),Event>

meaning that somebody (in our case, JOHN) or something or some event causes an event (in our case, the door to enter open state), and that this fact is an event. Here is the diagram, which again builds on the previous one:



Jackendoff's approach has been criticized for being underspecified when choosing functions. To quote Riemer [73, p. 269],

> the investigator simply looks for plausible underlying conceptual structures, but there are no clear procedures for determining when a primitive is justified…

Nevertheless, as a general method, it sheds light on how meaning can be formalized.

6.3.3 Generative Lexicon

The American linguist Pustejovsky [70] considers the problem of polysemy and asserts that, in many cases, what is considered as multiple senses of a given lexeme in dictionaries can be interpreted as the result of dynamical interaction between lexemes. For example, a dictionary could distinguish the senses of the verb "to bake" when applied to "a potato" (in which case the potato pre-exists as a natural object and only changes state from raw to baked) and when applied to "a cake" (in which case "cake" is an artifact that starts to exist by the act of baking ingredients such as flour, butter, cacao and the like). In Pustejovsky's approach, there is a single description of the verb "to bake," and its sense changes due to the inherent characteristics of the two objects: "potato" and "cake."

When Pustejovsky says "description" of the verb or the nouns, he means "representation in a formalism" so that his Generative Lexicon approach can be implemented (and indeed, it has been implemented).

How do we represent lexemes to obtain the desired behavior? Pustejovsky uses several description levels, the most important of which are: *event structure* (mainly for verbs: the nature of the events involved in the use of a given verb and their temporal order), *qualia structure* (a philosophical term that stands here for a structured set of descriptive characteristics) and *argument structure* (again for verbs: their valency and the semantic constraints applied to arguments, in terms of qualia).

Qualia structure can have four aspects: the *constitutive role* of the lexeme (its constituents or parts), its *formal role* (what distinguishes it in a larger domain, such as orientation, magnitude, shape, etc.), its *telic role* (from the Greek “τέλος”, *fulfillment*, the purpose and function of the object) and its *agentive role* (the origin or “bringing about” of the object).

We display on Table 6.2 the AVM representation of the verb “to build” to illustrate the Generative Lexicon approach [70, p. 82]. In the event structure, we have two

Table 6.2 The AVM representation of the verb “to build” [70, p. 82]

$$
\begin{bmatrix}
\textbf{build} & \\
\text{EVENTSTR} & \begin{bmatrix} \text{EVENT1} & e_1\text{:process} \\ \text{EVENT2} & e_2\text{:state} \\ \text{RESTR} & e_1 \le e_2 \end{bmatrix} \\
\text{ARGSTR} & \begin{bmatrix}
\text{ARG1} & \boxed{1} \begin{bmatrix} \textbf{animate-ind} & \\ \text{FORMAL} & \text{physobj} \end{bmatrix} \\
\text{ARG2} & \boxed{2} \begin{bmatrix} \textbf{artifact} & \\ \text{CONST} & \boxed{3} \\ \text{FORMAL} & \text{physobj} \end{bmatrix} \\
\text{D-ARG1} & \boxed{3} \begin{bmatrix} \textbf{material} & \\ \text{FORMAL} & \text{mass} \end{bmatrix}
\end{bmatrix} \\
\text{QUALIA} & \begin{bmatrix}
\textbf{create-lcp} & \\
\text{FORMAL} & \text{exist}\left(e_2, \boxed{2}\right) \\
\text{AGENTIVE} & \text{build-act}\left(e_1, \boxed{1}, \boxed{3}\right)
\end{bmatrix}
\end{bmatrix}
$$

possible events: e_1 is a process (the process of building), e_2 is a state (the one of a house being built). The event e_1 necessarily occurs before e_2. As for the argument structure, argument $\boxed{1}$ is the builder, an animate individual; argument $\boxed{2}$ is the object built, an artifact; $\boxed{3}$ is an optional argument referring to the material out of which $\boxed{2}$ is built. The link between $\boxed{2}$ and $\boxed{3}$ is in the constitutive quale (“quale” is the singular of “qualia”) of $\boxed{2}$, which is precisely $\boxed{3}$. Finally, the qualia structure belongs to a pattern labeled “create-lcp” (“lcp” stands for *lexical conceptual paradigm*, a way to characterize the lexical item). The formal quale denotes the existence of the built artifact $\boxed{2}$, and the agentive quale indicates how it came into existence: by being built by $\boxed{1}$ using $\boxed{3}$ as material.

Let us see now how the semantics of the VP “bake a cake” can be obtained from the semantics of “bake” and of “cake” [70, pp. 123–125].

We display in Table 6.3 the representation of the verb “to bake.” In other words, baking is a process where the baker is an animate individual, and the object being baked is a mass object. The lexical conceptual paradigm is one of changing state

Table 6.3 The AVM representation of the verb "to bake" [70, p. 123]

$$\begin{bmatrix} \textbf{bake} & \\ \text{EVENTSTR} & \begin{bmatrix}\text{E}_1 & e_1\text{:process}\end{bmatrix} \\ \text{ARGSTR} & \begin{bmatrix} \text{ARG1} & \boxed{1} \begin{bmatrix}\textbf{animate-ind} & \\ \text{FORMAL} & \text{physobj}\end{bmatrix} \\ \text{ARG2} & \boxed{2}\begin{bmatrix}\textbf{mass} & \\ \text{FORMAL} & \text{physobj}\end{bmatrix}\end{bmatrix} \\ \text{QUALIA} & \begin{bmatrix}\textbf{state-change-lcp} & \\ \text{AGENTIVE} & \text{bake-act}\left(e_1, \boxed{1}, \boxed{2}\right)\end{bmatrix}\end{bmatrix}$$

Table 6.4 The AVM representation of the noun "cake" [70, p. 123]

$$\begin{bmatrix} \textbf{cake} & \\ \text{ARGSTR} & \begin{bmatrix}\text{ARG1} & \text{x:food-ind} \\ \text{D-ARG1} & \text{y:mass}\end{bmatrix} \\ \text{QUALIA} & \begin{bmatrix}\text{CONST} & \text{y} \\ \text{FORMAL} & \text{x} \\ \text{TELIC} & \text{eat}\left(e_2, \text{z}, \text{x}\right) \\ \text{AGENTIVE} & \text{bake-act}\left(e_1, \text{w}, \text{y}\right)\end{bmatrix}\end{bmatrix}$$

(from raw to baked). And the agentive quale is the act of baking that takes three arguments: the process, the baker, and the object being baked.

We display in Table 6.4 the representation of the noun "cake." As we see, a cake is a "food-individual" created by a bake-act performed by some w and whose purpose is to be eaten by some z. If instead of a "cake," we had a "potato," then the agentive quale wouldn't be the bake-act since a potato is a root vegetable created by planting and growing. In the Generative Lexicon, there is a unification of qualia: "bake" and "cake" have the same agentive quale, and the composition of the two (Pustejovsky calls it a *co-composition*) is a VP that keeps the same agentive quale and acquires a new event: the one of the existence of the complement of the verb (see Table 6.5). As Pustejovsky notices [70, p. 125] the sense arises not by lexical enumeration from a dictionary but by co-composition of the verb and of the complement.

For example, Pustejovsky's approach has been applied in Artificial Intelligence to provide more thorough definitions of word senses in matching between an RDF-based lexicon and concepts in an ontology [45].

6.4 Cognitive Semantics

Cognitive semantics is a quite different thread of research in linguistics where some "sacred" structuralist principles about language are rejected. For example, language is not considered modular; therefore, the distinction between syntax and semantics is rejected, as well as between semantics and pragmatics.

Table 6.5 The AVM representation of the VP "bake a cake" [70, p. 125]

$$\begin{bmatrix} \textbf{bake a cake} & \\ \text{EVENTSTR} & \begin{bmatrix} \text{E1} & e_1\text{:process} \\ \text{E2} & e_2\text{:state} \\ \text{RESTR} & e_1 \leq e_2 \end{bmatrix} \\ \text{ARGSTR} & \begin{bmatrix} \text{ARG1} & \boxed{1} \begin{bmatrix} \textbf{animate-ind} & \\ \text{FORMAL} & \text{physobj} \end{bmatrix} \\ \text{ARG2} & \boxed{2} \begin{bmatrix} \textbf{artifact} & \\ \text{CONST} & \boxed{3} \\ \text{FORMAL} & \text{physobj} \end{bmatrix} \\ \text{D-ARG1} & \boxed{3} \begin{bmatrix} \textbf{material} & \\ \text{FORMAL} & \text{mass} \end{bmatrix} \end{bmatrix} \\ \text{QUALIA} & \begin{bmatrix} \textbf{create-lcp} & \\ \text{FORMAL} & \text{exist}\left(e_2, \boxed{2}\right) \\ \text{AGENTIVE} & \text{bake-act}\left(e_1, \boxed{1}, \boxed{3}\right) \end{bmatrix} \end{bmatrix}$$

An important concept in cognitive semantics is the concept of concept (pun!). The study of linguistic meaning is considered a study of conceptual structure in the sense of "thoughts, concepts, perceptions, images, and mental experience in general" [73, p. 239]. There is no distinction between "linguistic knowledge" and "encyclopedic knowledge," any aspect of the latter is considered in the linguistic study process.

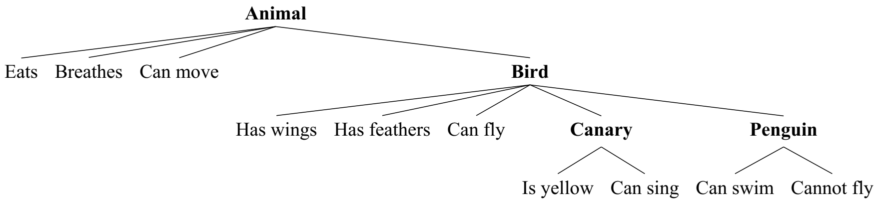
6.4.1 Prototype Theory

> los animales se dividen en: (a) pertenecientes al Emperador, (b) embalsamados, (c) amaestrados, (d) lechones, (e) sirenas, (f) fabulosos, (g) perros sueltos, (h) incluidos en esta clasificación, (i) que se agitan como locos, (j) innumerables, (k) dibujados con un pincel finísimo de pelo de camello, (l) etcétera, (m) que acaban de romper el jarrón, (n) que de lejos parecen moscas.[3]
>
> Jorge Luis Borges, "El idioma analítico de John Wilkins," *Otras inquisiciones*

In componential analysis, features are crisp: an alligator is definitely dangerous, and a shark is indisputably aquatic. But how does the human brain store and process these features? According to Quillian [71], we store information in *semantic networks*, i.e.,

[3] "animals are divided into (a) those that belong to the emperor; (b) embalmed ones; (c) those that are trained; (d) suckling pigs; (e) mermaids; (f) fabulous ones; (g) stray dogs; (h) those that are included in this classification; (i) those that tremble as if they were mad; (j) innumerable ones; (k) those drawn with a very fine camel's-hair brush; (l) etcetera; (m) those that have just broken the flower vase; (n) those that at a distance resemble flies."

rooted trees, the leaves of which are features, and all the other nodes are categories. Here is an example of such a tree (categories are in bold):



This tree has an inheritance property (i.e., features of ancestors apply to descendants) unless otherwise stated (i.e., penguins cannot fly). In a study by Quillian [12], questions were asked to participants, and their answers were timed. It came out that participants tend to answer much quicker when a feature is characteristic of a concept (such as "Do canaries sing?") instead of when it is a feature of a concept higher in the hierarchy (such as "Do canaries breathe?"), which requires inference.

Let us now reverse the situation: Instead of checking whether a given concept has specific features (inherited or not), we can ask the question: "How do we classify instances of entities into categories?", i.e., how does the psychological process of categorization function? In our example, are birds those animals that have wings and feathers and can fly? How certain can we be when asked if a specific animal is a bird, knowing there are quite a few exceptions? (Squirrels can fly, bats have wings, dinosaurs apparently had feathers…)

Studies by Rosch [76, 75] have shown that we use a *typicality gradient*: we place members of a category on a scale of typicality, e.g., robins are more typical members of the bird category than chicken. The list of birds, ordered by decreasing *birdiness* is: robins, eagles, chickens, ducks, geese, penguins, pelicans, bats [47, p. 459]. Similarly, apples are more typical fruits than watermelons. The study even showed that, in the low typicality range, people often make false assumptions about membership to categories, e.g., many participants in an experimentation classified a stroke as being a disease and pumpkins as being vegetables [59]. Incidentally, as the title of McCloskey's study "Natural categories: Well-defined or fuzzy sets?" [59] implies (and as Lakoff already emphasized in 1973 [47]), a convenient mathematical formalism for the cognitive categorization process is *fuzzy set theory*, as introduced by Zadeh in 1965 [100].

From typicality, it is only one step to move to *prototypicality*, namely the property of being among the most typical members of a category. How can we identify prototypical members of a given concept in linguistic data (without going through psycholinguistic experimentations)? Rosch [77, p. 39] provides two methods:

Hedges. Hedges are words or phrases expressing ambiguity, probability, caution, or indecisiveness. When a statement "x is a X" uses a hedge, it cannot be prototypical. Compare "a chicken is technically a bird" with "a canary is technically a bird": the latter sounds unnatural since it is obvious that a canary is a bird. At the same time, the former underlies the non-prototypicality of "chicken" in the "bird" category. Lakoff [47, p. 472] gives a list of hedges, including the "x

is the y of Z" pattern, illustrated by the phrase "Chomsky is the De Gaulle of Linguistics."

Hedges are a very interesting phenomenon because they can alter meaning significantly. Lakoff [47, p. 474] gives the following example:

> Esther Williams is a fish.
>
> Esther Williams is a regular fish.

Considering that Esther Williams was a competitive swimmer in the 1930s, the first sentence (without hedge) is unnatural since it is evident that E.W. is not a fish. The hedge "regular" in the second sentence calls for a different interpretation: E.W. swims well and is at home in the water. Regarding category membership, the hedge "regular" implies that E.W. is not a member of the biological category of fishes but a member of the category of entities with fish-like behavior and/or competencies. Ergo, E.W. is not a prototypical fish.

Substitutability into sentences. Take a direct hyperonym w' of a word w, take sentences using w', and check to what extent w' can be substituted by w in the sentences. This is one of the funniest linguistic tests ever. As an example, take the verse of a poem by Adoni Marcano:

> I love the sound of birds
> so early in the morn

Replacing "birds" with its hyponym "chicken," the poem loses all its charm.[4] This shows that chickens are not prototypical birds.

6.4.2 Fillmore's Frames

In the early seventies, many people independently introduced similar notions, namely "scene/frame," "schema," "script," and "structure of expectation." Fillmore [19] introduced a dual structure: on the one hand, a description of a situation with identification of agents and entities and the role they play in the situation—this he calls *scene*; and on the other hand, a linguistic analysis of the situation, showing how expressions and grammatical patterns highlight aspects of it—this he calls *frame*. Later, he abandoned the term "scene" and used only "frame." In [21], he gave a nice description of the concept of "frame":

> Speakers can be said to know the meaning of the word only by first understanding the background frames that motivate the concept that the word encodes. Within such an approach,

[4] Appreciation of bird calls can be very subjective. Take the example of the Irish *corncrake* `https://youtu.be/KpEqHGygCbg` that sounds like a squeaky door handle. Still, the author would love to be in an environment where corncrakes can be heard "in the morn," so the general ambiance can be just as important as the physical characteristics of the corncrake's call. Independently of that, and besides its unique vocal signature, the corncrake is interesting from a linguistic perspective because it is one of the few species having an onomatopoetic Latin name, namely *crex crex*, which proves that Linnaeus, who named the species in 1758 [53, p. 153], already appreciated the corncrake's sound for its true worth.

words or word senses are not related to each other directly, word to word, but only by way of their links to common background frames and indications of the manner in which their meanings highlight particular elements of such frames.

In the same paper, Fillmore provides a (simplified) example of frame: he chooses the subject of "commercial transaction" and, more specifically, a scenario in which an agent *A* acquires control or possession of an entity from an agent *B*, by agreement (hence it cannot be theft) and as the result of surrendering a sum of money to agent *B*. The background requires understanding property ownership, money economy, implicit contracts, and more. The categories that can be used in the commercial transaction frame are Buyer, Seller, Goods, and Money. Considering the category Money, he distinguishes the monetary value of the goods as determined by the Seller (or negotiated between Seller and Buyer) from the amount of money the Buyer transmits to the Seller. He derives the secondary categories Cost, Tender, and Charge from this. Further elaborations of the scenario, for structurally more complex members of the lexical field, have to do with negotiations of the cost ("bargaining," "discount"), differences between the time of the Goods-transfer and the time of the Money-transfer ("credit"), the possibility of total or piecemeal transfer of money ("time payment"), the difference between price and value, the difference between the value of the tender and the actual physical currency that is tendered, and so on. This is the "encyclopedic" part of the frame (previously called "scene").

On the linguistic side, he builds the following table:

	Buyer	Seller	Goods	Money
BUY	Subject	(from-PP)?	Direct object	(for-PP)?
SELL	(to+pp)?	Subject	Direct object	(for-PP)?
CHARGE	(Indirect object)?	Subject	(for-PP)?	Direct object
SPEND	Subject	NULL	for/on-PP	Direct object
PAY	Subject	[Indirect object]?	[for-PP]?	Direct object
PAY	Subject	(to-PP)?	for-PP	Direct object
COST	(Indirect object)?	NULL	Subject	Direct object

In this table, parentheses and question marks denote optionality, square brackets denote omissibility under conditions of anaphora (when the identity of the object representing the category is given in the context), and foo-PP denotes a PP headed by preposition foo. In the example "Harry spent twenty dollars on a new tie," "Harry" is the Buyer, "twenty dollars" is the Money, the optional mention of the Goods is marked with the preposition "on," and the Seller cannot be introduced because of the NULL entry in the table.

6.4.3 FrameNet

In their 1992 paper [21], Fillmore & Atkins wrote:

The kind of description we seek to justify could not easily be represented in a standard print dictionary, for reasons that soon become clear, but we imagine, for some distant future, an

> online lexical resource, which we can refer to as a "frame-based" dictionary, which will be adequate to our aims. In such a dictionary (housed on a workstation with multiple windowing capabilities), individual word senses, relationships among the senses of polysemous words, and relationships between (senses of) semantically related words will be linked with the cognitive structures (or "frames"), knowledge of which is presupposed for the concepts encoded by the words.

The "distant future" finally wasn't that distant after all: in 1997, the project Fillmore & Atkins imagined became a reality. As of today (May 2023) it[5] contains 1,224 frames with 1,878 frame relations. The frames contain 13,687 lexical units, of which 5,575 are nouns, 5,214 are verbs, and 2,407 are adjectives. FrameNet is an invaluable resource for NLP since it provides the background knowledge an AI tool may need to realize the ins and outs of most situations. It also allows it to communicate with humans in full knowledge of the facts.

In the following, we give a small but complete example: the FrameNet frame of `commercial_transaction`. On the Web page, some words are displayed with a colored background. We will use brackets instead, followed by the corresponding frame element in the index position. The frame has four parts: `Definition`, `Frame Elements`, `Frame-Frame Relations`, and `Lexical Units`.

The first part briefly and concisely describes the frame situation and gives an example. Uppercased words denote lexemes ("lexical units" in this context). The word `TRANSACTION` is the target lexical unit of the frame.

Definition:

These are words that describe basic commercial transactions involving a $[\texttt{Buyer}]_{\texttt{Buyer}}$ and a $[\texttt{Seller}]_{\texttt{Seller}}$ who exchange $[\texttt{Money}]_{\texttt{Money}}$ and $[\texttt{Goods}]_{\texttt{Goods}}$. The individual words vary in the frame element realization patterns. For example, the typical patterns for the verbs buy and sell are: BUYER buys GOODS from the SELLER for MONEY. SELLER sells GOODS to the BUYER for MONEY.

> $[\texttt{His}]_{\texttt{Seller}}$ $[\texttt{\$20}]_{\texttt{Money}}$ $[\texttt{TRANSACTION}]^{\texttt{Target}}$ $[\texttt{with Amazon.com}]_{\texttt{Seller}}$ $[\texttt{for a new TV}]_{\texttt{Goods}}$ had been very smooth.

The second part lists core and non-core semantic roles (frame elements). An abbreviation and a gloss follow each frame element.

Frame Elements:

Core:

$[\texttt{Buyer [Byr]}]_{\texttt{Buyer}}$ The $[\texttt{Buyer}]_{\texttt{Buyer}}$ wants the $[\texttt{Goods}]_{\texttt{Goods}}$ and offers $[\texttt{Money}]_{\texttt{Money}}$ to a $[\texttt{Seller}]_{\texttt{Seller}}$ in exchange for them.

$[\texttt{Goods [Gds]}]_{\texttt{Goods}}$ The Frame Element Goods is anything (including labor or time, for example) that is exchanged for Money in a transaction.

$[\texttt{Money [Mny]}]_{\texttt{Money}}$ Money is the thing given in exchange for Goods in a transaction.

$[\texttt{Seller [Slr]}]_{\texttt{Seller}}$ The $[\texttt{Seller}]_{\texttt{Seller}}$ has possession of the $[\texttt{Goods}]_{\texttt{Goods}}$ and exchanges them for $[\texttt{Money}]_{\texttt{Money}}$ from a $[\texttt{Buyer}]_{\texttt{Buyer}}$.

Non-Core:

[5] https://framenet.icsi.berkeley.edu/fndrupal/

[Means [Mns]]$_{\text{Means}}$ (Semantic Type: State_of_affairs) How a commercial transaction occurs.

[Rate [Rate]]$_{\text{Rate}}$ Price or payment per unit of Goods.

[Unit [Unit]]$_{\text{Unit}}$ The Unit of measure of the Goods according to which the exchange value of the Goods (or services) is set. Generally, it occurs in a by-PP.

The third part lists frame-to-frame links. Reciprocality is the parent frame in the sense of the is-a relation. The Subframe relation is not an is-a relation but a part-of relation in a given order. To give an example from a different frame than `Commercial_Transaction`, the `Criminal_process` frame has the Subframes `Arrest`, `Arraignment`, `Trial Sentencing`, `Appeal`.

Let us continue exploring the `Commercial_Transaction` frame:

Frame-frame Relations:

Inherits from: Reciprocality

Subframe of: Commerce_scenario

Has Subframe(s): Commerce_goods-transfer, Commerce_money-transfer

Finally, the last part contains the lexical unit paired with the frame:

Lexical Units:

transaction.n

where *n* stands for "noun". In the "Lexical Entry" page dedicated to this lexical unit, we find the following information:

Lexical Entry

transaction.n

Frame: Commercial_transaction

Definition:

COD: an instance of buying or selling.

Frame Elements and Their Syntactic Realizations

The Frame Elements for this word sense are (with realizations):

Frame Element	Number Annotated	Realization(s)
[Buyer]$_{\text{Buyer}}$	(5)	INI.-- (4) Poss.Gen (1)
[Goods]$_{\text{Goods}}$	(5)	INI.-- (4) PP[of].Dep (1)
[Money]$_{\text{Money}}$	(5)	INI.-- (5)
[Seller]$_{\text{Seller}}$	(5)	INI.-- (5)

Valence Patterns:

These frame elements occur in the following syntactic patterns:

Number Annotated	Patterns			
5 TOTAL	[Buyer]$_{\text{Buyer}}$	[Goods]$_{\text{Goods}}$	[Money]$_{\text{Money}}$	[Seller]$_{\text{Seller}}$
(3)	INI --	INI --	INI --	INI --
(1)	INI --	PP[of] Dep	INI --	INI --
(1)	Poss Gen	INI --	INI --	INI --

These tables inform us that the lexical unit `Transaction` can be combined with the lexical unit of the `Buyer`, using possessive genitive case (e.g., “Elon Musk’s transaction”) and with the lexical unit of Goods, in a prepositional phrase using the preposition “of” (e.g., “a transaction of peanuts”). The numbers in parentheses are links to annotated examples at the bottom of the screen.

More information on FrameNet can be found in [80], freely available on the project’s Web site.

6.4.4 Minsky’s Frames

In 1975, inspired by Fillmore’s 1968 paper [20], the American computer scientist Minsky, one of the pioneers of AI (and a consultant of Kubrick for *2001: A Space Odyssey*), wrote a paper that became seminal in AI [60]. In this paper, he describes his vision of frames in artificial intelligence.

A *Minsky frame*, as defined in [60], is a graph, the nodes of which can hold various kinds of information about a given situation. Quoting Minsky [60], the goal is “how to use the frame or what one can expect to happen next.” Minsky distinguishes upper and lower levels of the graph: the top levels contain “fixed things that are always true,” and the lower level has slots that specific data instances must fill. He calls them *terminals*. Terminals can have type constraints called “markers” and default values “attached loosely to their terminals so that they can be easily displaced by new items that fit better the current situation.”

The first frame example he gives is about computer vision: a frame captures information on the three visible sides of a cube; the cube turns, and new frames are created so that the system builds a 3D representation of the cube in memory.

Then he moves to linguistics with a critique of Chomsky’s famous sentence, “colorless green ideas sleep furiously.” He does admit that syntax matters and that a sentence with the same words ordered differently would be meaningless. Still, he disagrees with Chomsky that the “logical meaningless” of the specific sentence is psychologically void. He describes a possible frame for the sentence:

> The sentence [“colorless green ideas sleep furiously”] can certainly generate an image! The dominant frame is perhaps that of someone sleeping; the default system assigns a particular bed, and in it lies a mummy-like shape-frame with a translucent green color property. In this frame there is a terminal for the character of the sleep—restless, perhaps—and “furiously” seems somewhat inappropriate at that terminal, perhaps because the terminal does not like to accept anything so “intentional” for a sleeper. “Idea” is even more disturbing, because one expects a person, or at least something animate. One senses frustrated procedures trying to resolve these tensions and conflicts more properly, here or there, into the sleeping framework that has been evoked. [60, p. 109]

Undoubtedly, Minsky could author a horror science fiction movie scenario.[6]

He then draws a parallel between the top levels of the frame and syntax and between lower levels and semantics:

[6] And indeed he is the co-author, with Harrison, not of a horror movie scenario but of a technological science fiction novel *The Turing Option* (1992), see also p. 190.

> if the top levels are satisfied but some lower terminals are not we have a meaningless sentence; if the top is weak but the bottom solid, we can have an agrammatical but meaningful utterance. [60, p. 109]

Finally, he draws a parallel between prototypes in semantic networks and (what is called today) *hub-and-spoke* optimization of air traffic networks [63, p. 479]: to travel from village A in one country to village B in another country, we drive from A to a larger town A' having an airport and from there we fly to an even larger town A'' with an international airport, and from there to a large town B'' in the destination country, and then we fly to a smaller town B', and finally we drive to village B. In his metaphor, countries are categories, larger towns are prototypes, smaller towns are prototypes of subcategories, and so on.

☞ 297 Minsky's frames inspired frame-based systems, which, in turn, evolved into the object-oriented programming paradigm [72], which is still predominant today in the world of software. Furthermore, among the frame-based systems inspired by Minsky's frames was also KL-ONE [18], which was the ancestor of description logics [62] used nowadays in the Semantic Web.

6.4.5 Frames and Humor

Linguistics is a dead serious science, so it may come as a surprise that there are indeed linguists specialized in… humor (see, e.g., Coulson [13], Giora [27], and Brône *et al.* [6]). We digress on humor here because many jokes are based on *frame-shifting*. As Minski observed already in 1984:

> The element that seems to me most common to all the different kinds of humor is that of unexpected frame-substitution, in which a scene is first described from one viewpoint and then suddenly—typically by a single word—one is made to view all the scene-elements in another, quite different way. [61, p. 10]

According to Giora,

> To lead our "one-track mind" down "the garden path," the initial context of a joke is usually unambiguous, compatible with the salient meaning, so that this interpretation is retained up until the punch line, at which point a sudden incongruity forces reinterpretation. [27, p. 168]

Interpretation uses frames, and re-interpretation results in a frameshift. Quoting Ritchie,

> Humor and irony are usually based on activation of alternative frames, a socially-licensed, dominant frame that is consistent with conventional social norms and a "subversive" frame that questions or negates key features of the socially approved frame. [74, p. 27]

Let us take a closer look at four jokes by Groucho Marx. In

> She got her good looks from her father. He's a plastic surgeon

the first sentence uses a frame on family, with a slot on heredity and resemblance between family members. The second sentence does not contradict the good looks or the resemblance with the father. Still, it makes these elements irrelevant since we shift to a completely different frame, namely the one of plastic surgery. Here, the subversive element is human intervention in somebody's appearance.

The second joke,

> With a little study you'll go a long ways, and I wish you'd start now

builds on the standard pattern of a moral quote. The frame we enter is one of self-improvement and professional success. It uses the metaphor of life being a journey and success being proportional to the distance that journey covers. The second clause (using the appropriate prosody) de-activates the metaphor. It shifts the frame to the immediate physical distance between speaker and listener, the former wishing the latter to leave. The subversive element is the sudden and implicit aggressiveness.

In the joke

> You know you've got the brain of a four-year-old child, and I bet he was glad to get rid of it

frame-shifting uses the shift from the idiomatic meaning of "having the brain of" to its literal meaning. Common usage of the idiom refers to the meaning "having the level of intelligence of," while the second clause shifts from the frame of intelligence level to the one of actually holding the physical brain of a child, therefore being a monstrous child murderer (as in the *Silence of the Lambs* movie). Still, the macabre effect is quickly softened by a second frame shift: the one of the brain being an unnecessary and probably malfunctioning organ that the child is supposedly glad to get rid of.

Finally, the joke

> If you want to see a comic strip, you should see me in the shower

uses part-of-speech and syntactic ambiguity: at first, the reader considers "comic strip" as an adjective followed by a noun, and the appropriate frame is the one of the reading act of comic strips in newspapers or magazines. The second clause forces the reader to re-parse the first clause and to consider "comic" as a noun (referring to the narrator) followed by the verb "to strip." The frameshift is dramatic: from the (positively loaded) mental image of a cartoon, we move to the one of the narrator (Groucho Marx) naked in his shower. The subversive frame is the one of intimacy and its violation.

Another much more recent frame-shifting joke is the bar-rabbit joke

> A priest, a rabbit, and a minister walk into a bar. The rabbit says, "I might be a typo."

From the usual bar-related situational frame, we shift to a frame of characters knowing they are parts of a written joke, of which the writing instrument can (mistakenly) change the character distribution [1].

6.4.6 Idealized Cognitive Models and Conceptual Theory of Metaphor

Chomsky is the De Gaulle of Linguistics

George Lakoff, *A study in meaning criteria…*

According to Lakoff, the Australian aboriginal language Dyirbal has a category that consists of

> women, fire, and dangerous things, but also the platypus, the bandicoot, and the echidna. [48, p. 5]

This Dyirbal category was his inspiration for the—today probably considered as being politically incorrect—title *Women, Fire, and Dangerous Things* of [48], even though there is a historical very similar maxim by the Greek theatrical author Menander (343–291 BCE), that goes like "Θάλασσα καὶ πῦρ καὶ γυνὴ τρίτον κακόν"$_{EL}$ ⌊After sea and fire, a woman is the third evil⌋ [14, p. 94].

In this very book, Lakoff introduced the concept of *idealized cognitive models* (ICM), which is an extended version of a frame. He defines

- *propositional* ICMs (which he considers as being Fillmore's frames), but also
- *image-schematic* ICMs (that specify images, trajectories, shapes, containers, etc.),
- *metaphoric* ICMs (that are mappings from a propositional or image-schematic ICM in one domain to a corresponding structure in some other domain) and
- *metonymic* ICMs (propositional or image-schematic ICMs together with a function from one model element to another, for example, a function that enables the part to stand for the whole).

Lakoff, together with Johnson, has also introduced *Conceptual Metaphor Theory* through a very influential book, *Metaphors We Live By* [49], published in 1980.

The most important idea of the book is perfectly summarized in its first paragraph:

> Metaphor is for most people a device of the poetic imagination and the rhetorical flourish—a matter of extraordinary rather than ordinary language. Moreover, metaphor is typically viewed as characteristic of language alone, a matter of words rather than through or action. For this reason, most people think they can get along perfectly well without metaphor. We have found, on the contrary, that metaphor is pervasive in every day life, not just in language but in thought and action. Our ordinary conceptual system, in terms of which we both think and act, is fundamentally metaphorical in nature.

In other words, our way of living in the world and interacting with it is metaphoric. Let us take the following example of common sentences using the same metaphoric pattern (set in small caps):

LOVE IS A JOURNEY

Look *how far we've come*.
We're *at a crossroads*.
We'll just have to *go our separate ways*.
We can't *turn back now*.
I don't think this relationship is *going anywhere*.

> *Where* are we?
> We're *stuck*.
> It's been a *long, bumpy road*.
> This relationship is a *dead-end street*.
> We're just *spinning our wheels*.
> Our marriage is *on the rocks*.
> We've gotten *off the track*.
> This relationship is *foundering*. [49, pp. 44–45]

The argument is that it is not our poetic creativity that has created all these expressions but rather the fact that we indeed perceive a romantic relationship as a journey, and even a pretty abstract journey since, as Lakoff & Johnson remark, these expressions refer to three different types of travel: by *car* ("bumpy road"), by *train* ("off the tracks") and by *boat* ("on the rocks").

Furthermore, Lakoff & Johnson remark that metaphors are open-ended, contrary to idioms, so we can further develop them if we respect the mapping from domain A to domain B. For example, to stay within the LOVE IS A JOURNEY example, we can continue producing expressions such as "We are not going with the same speed," "You're leaving me behind," "We have a long road ahead"… Not to mention the many non-linguistic metaphors that exist, like the thumb down gesture, which is an *orientational metaphor*. The LOVE IS A JOURNEY metaphor is called *structural*, there is a third type: *ontological metaphors*, such as personification: "*Life has cheated* me," "This *fact argues* against the standard theories," etc.

Geeraerts [25] criticizes *Conceptual Metaphor Theory* by arguing that there exist *dead metaphors*, i.e., expressions that have been metaphors at some time but now have lost their metaphoric character. The example given is the verb "to strike": the sentence "His sincerity struck me," is not anymore an illustration of the metaphor EMOTIONAL EFFECT IS PHYSICAL CONTACT—instead, the verb has acquired a meaning "to surprise, to affect suddenly".

6.4.7 MetaNet

The *MetaNet* corpus was created in 2013 [36] and is hosted at the International Computer Science Institute in Berkeley, California. It contains (as of November 2022) 685 metaphors based on 576 frames. We wanted to illustrate it by the example LOVE IS A JOURNEY and indeed it contains a metaphor A ROMANTIC RELATIONSHIP IS A JOURNEY. Still, unfortunately, even though the metaphor is there, the two frames Romantic Relationship and Journey are missing.

We will therefore present a different metaphor: KNOWING IS TOUCHING, as in John 20:25,

> Except I shall […] put my finger into the print of the nails, and thrust my hand into his side, I will not believe.

The frames of this metaphor are `Touching` and `Knowing`. Here is their description:

```
Roles:
Role Name: toucher
```

```
Role Name: touched_thing
Role Name: sense_of_touch
Role Name: touch_input
Role Name: touch_bodypart
Definition/Comments: part of body that contacts/touches another entity
```

Related Frames:

```
Current Frame: Touching
Relation Type: is subcase of
Related Frame: Perception

Current Frame: Touching
Relation Type: makes use of
Related Frame: Contact
```

Bindings:

```
toucher=Perception.perceiver
touched_thing=Perception.perceived_thing
toucher=Contact.entity1
touched_thing=Contact.entity2
```

Relevant Lexical Units:

```
Lempos touch.v
Lempos touch.n
```

As we see, `Touching` inherits from `Perception`, and the roles `toucher` and `touched_thing` are bound with the roles `Perception.perceiver` and `Perception.perceived_thing`. The `Knowing` frame is defined similarly:

Roles:

```
Role Name: knower
Role Name: knowledge
Definition/Comments: conceptual content
Role Name: knowing_x-schema
Role Name: knowledge_capacity
Definition/Comments: the knower's ability to acquire and retain knowledge
Role Name: knowledge_status
Definition/Comments: whether or not conceptual content is known to knower
```

Related Frames:

```
Current Frame: Knowing
Relation Type: is subcase of
Related Frame: Cognizing
```

Bindings:

```
knower=Cognizing.cognizer
knowledge=Cognizing.content
```

`Knowing` inherits from `Cognizing` and, once again, some roles of the former are bound with roles of the latter. The KNOWING IS TOUCHING metaphor will now join the two frames as follows:

Source Frame: Touching
Target Frame: Knowing

```
Mappings:
knower <= toucher
knowing_x-schema <= touching_x-schema
mind <= limbs
idea <= object
```

This is the most important information: the metaphor is oriented from `Touching` to `Knowing`, and therefore the mappings will be in this direction: `toucher` → `knower`, `touching_x-schema` → `knowing_x-schema`, but also lexical units that do not appear in the frames: `limbs` → `mind` and `object` → `idea`. The description continues with the relationship of this metaphor with three other metaphors:

```
Related Metaphors:
Current Metaphor KNOWING IS TOUCHING
Relation Type has as transitive subpart 1
Related Metaphor SEEING IS TOUCHING

Current Metaphor KNOWING IS TOUCHING
Relation Type has as transitive subpart 2
Related Metaphor KNOWING IS SEEING

Current Metaphor KNOWING IS TOUCHING
Relation Type is a source subcase of
Related Metaphor KNOWING IS PERCEIVING
```

According to MetaNet, KNOWING IS TOUCHING can be decomposed into KNOWING IS SEEING and SEEING IS TOUCHING. Also, by replacing the target frame with its parent, we get the metaphor KNOWING IS PERCEIVING. The description ends with two examples:

```
Examples:
Example Text: I've almost got an idea, but I can't quite put my finger on
it.
Example Text: I could never put my finger on what the problem was.
```

There is currently a lot of research (see, e.g., Veale *et al.* [94]) on metaphor detection and annotation, as well as on the effect of metaphor in knowledge extraction.

6.5 Formal Semantics

6.5.1 Frege, Sense, Denotation, and Truth

Much of what we will see in this section originates from the books and lectures of the German philosopher Frege (1848–1925). Shy and introverted (he gave his lectures turning his back to students, something no teacher would do today, be it for security reasons), Frege started analyzing mathematical formulas and then applied his findings to language in general. Take symbols like "π," "0," "$\frac{1}{2}$," that *denote* abstract mathematical objects. He observed that sometimes there are several ways of denoting the same object, as in "3 + 2" and "5," but even though they denote the same object, we do not necessarily perceive them in the same way. He called this perception, or more precisely, the object's "cognitive significance" [102], its *sense*. So we say that "3 + 2" and "5" have different *senses* but the same *denotation* (some translators used the term *reference* instead of *denotation*). He then noticed that when we write "3 + 2," it is only a special case of a placeholder "() + ()," where we have assigned numbers to the two empty locations. The sum is a binary function in that it has two placeholders, and once values have been provided, it returns a new denotation, namely the one of the value of the sum of the two numbers. Objects such as numbers or elements of mathematical structures have their denotations, but when they are used as arguments of functions, they result in new denotations. So what is the denotation of a formula such as "1 + 1 = 2"? In mathematics, the answer to this question seems trivial: the only thing we can say about this formula is whether it is true or false: "1 + 1 = 2" is true (in all groups besides $\mathbb{Z}/2\mathbb{Z}$, where there is no "2" symbol), while "1 + 1 = 3" is false. And therefore, Frege decided that the denotation of a formula would be its *truth value*. This is a revolutionary idea since "truth" is not a mathematical object: it is neither a number, nor an algebraic structure, nor a geometric figure, but an abstract concept. Nevertheless, we do understand Frege's motivation for taking this decision: after all, all that can be said about the formula "1 + 1 = 2" is whether it is true or false.

He then applied the same approach to language and the real world. He took the example of the "Morning Star" and the "Evening Star": the senses of these terms are different, but their denotation is the same (namely Venus, which is not even a star but a planet). Functions match objects of the real world with other objects of the real world; for example, the father of Luke Skywalker is Darth Vader, and therefore, we move from the denotation of "Luke Skywalker's father" to the denotation of "Darth Vader." A "formula" in this case would be a proposition such as "Darth Vader is the father of Luke Skywalker." Following Frege's logic, the denotation of this proposition would be a *truth value*, its truth value in a given context.

☞ 277 Does this ring a bell? It is exactly how First-Order Logic represents information and knowledge about the world. What Frege calls "sense" is a logical term, and what he calls "denotation" is the interpretation of the term in a given domain. His "functions" are also called functions in FOL, and his formulas (which he calls "concepts," but let us not stick too much to his terminology) are FOL predicates.

Before closing this section on Frege and truth, a small side note: the reader should not imagine that "truth" is a trivial, unproblematic concept. A paradox known since Antiquity called the "Liar Paradox" goes as follows: imagine an auto-referential sentence saying, "This sentence is false." If it is false, the sentence must be true, but in this case, it must be false. This may seem like a joke, but the great logician Tarski took it very seriously:

> In my judgment, it would be quite wrong and dangerous from the standpoint of scientific progress to depreciate the importance of this and other antinomies, and to treat them as jokes or sophistries. It is a fact that we are here in the presence of an absurdity, that we have been compelled to assert a false sentence. … If we take our work seriously, we cannot be reconciled to this fact. [90, p. 348]

Tarski attempted to solve the problem by claiming that for a language to be consistent, its sentences should not make claims *about other sentences*. If necessary, one should use a higher-level language instead, and another for making claims on the higher-level sentences, and so on. This idea of a hierarchy of languages was turned down by another logician, Saul Kripke, who invented the paradox

> for every level-σ statement in the language there is a level-$(\sigma + 1)$ statement saying that the former is wrong,

which is a statement about all levels of the language, and hence belongs to no level… [41] Another attractive "solution" to the liar paradox was given by Zadeh, the inventor of fuzzy logic, whose proposal is that the fuzzy truth value of the statement "this sentence is false" shall be 0.5, as a solution of the equation $x = 1 - x$ [101, 33].

6.5.2 Montague Formal Semantics

Montague, a student of Tarski, introduced a new approach to the semantics of natural languages based on First-Order Logic. His approach builds on Chomsky's generative grammar. He claims that for every Chomskian rewrite rule, we have a converse semantic rule so that if we can represent the semantics of the leaves of a syntax tree in some way, then we can apply the converse rules and move up to the root of the tree, thereby obtaining the semantics of the sentence. ☞ 277

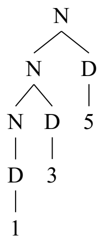
In some sense, this is an "implementation" of the compositionality principle: not only does Montague claim that the meaning of a sentence can be obtained from the meanings of its elementary units and from the way these are combined, but he gives a concrete method for obtaining the meaning of the sentence out of the meanings of elementary units and the syntax tree of the sentence. ☞ 5

This happens, for example, when we use a calculator to write integer numbers. Let us take the example of a simple calculator to illustrate Montague's approach. Consider that the "meaning" (the "denotation" in Frege's vocabulary) of a digit is its *numerical value*: the meaning of "5" (or "V" in Roman, or "ε′"$_{\text{EL}}$ in Greek, or "ה"$_{\text{HE}}$ in Hebrew, or "五"$_{\text{ZH}}$ in Chinese) is the numerical value five, the number of fingers most people have in a single hand. How do we get the numeric value of a multi-digit number?

This will be our first compositionality challenge: knowing the "meaning" of "1," "3," and "5," how do we get the "meaning" of "135"? Let us write down the syntax rules of the decimal notation of integer numbers (i.e., let us write down the grammar of the formal language of alphabet $\{0, 1, \ldots, 9\}$, the words of which are integer numbers): ☞ 247

(s_1) $\quad \mathrm{N} \to \mathrm{N}\ \mathrm{D}$
(s_2) $\quad \mathrm{N} \to \mathrm{D}$
(s_3) $\quad \mathrm{D} \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$

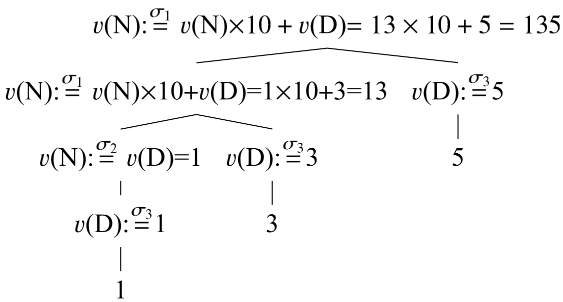
This means that a number can be rewritten as a number followed by a digit or as a digit. Digits belong to the set of symbols $\{0, 1, \ldots, 9\}$. The syntax tree of "135," therefore, is



In this tree, we have applied syntactic rule s_1 twice, rule s_2 once and rule s_3 thrice. What will the corresponding semantic rules be? Let us denote by $v(.)$ the "meaning," that is, the numeric value of any node of this tree. The semantic rules will allow us to move upwards and calculate the meaning of a node out of the meanings of its children:

(σ_1)	$v(\mathrm{N}) := 10 \times v(\mathrm{N}) + v(\mathrm{D})$
(σ_2)	$v(\mathrm{N}) := v(\mathrm{D})$
(σ_3)	$v(\mathrm{D}) := i$, for $i \in \{0, 1, \ldots, 9\}$.

Rule σ_1 says that "if you add a digit to the right of a number, then the new value is ten times the value of the number plus the value of the digit," which is the basic principle of the Arabic, Hebrew[7] and Indian numbering systems. We can now calculate the "meaning" of each node and annotate the syntax tree accordingly:



Montague applied the same approach to natural language, where the "meaning" is represented in first-order logic. So the question is: how do we describe the meaning of individual words in FOL, and what are the semantic counterparts of the various syntactic rules? Let us take the example of the sentence "Clara sleeps." We can already state two things:

(1) we can assign to the word "Clara" the meaning `Clara`, considered as a FOL constant,
(2) the meaning of the sentence "Clara sleeps" is given by the FOL formula `sleeps(Clara)`.

But what about the word "sleeps"?

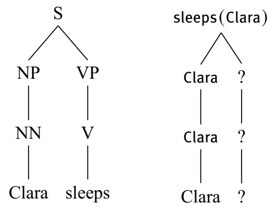
Here are the syntax rules necessary to obtain the syntax tree of the sentence:

---

[7] With an exception for Hebrew: numbers 15 and 16 are written טו and טז respectively, to avoid writing the name of God.

(s_1) S → NP VP
(s_2) NP → NN
(s_3) VP → V
(s_4) NN → Clara
(s_5) V → sleeps

and the syntax tree will be

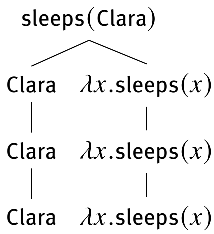


where we have attempted to draw, on the right, the tree of meanings (with question marks for vertices that we don't know yet how to represent).

In logical expressions, predicates with arguments are always written with their arguments: we can write sleeps(Clara) or sleeps(X) where X is some variable, or even sleeps(father(cousin(best-friend(Amy)))), but never "sleeps( )" or "sleeps(.)." This is why Montague uses not plain FOL but FOL enhanced by the λ-calculus formalism. This formalism, introduced by Church in the thirties, is a convenient way of defining functions without naming them. Its main ingredient is the λ operator, which is always followed by some variable, a period, and a FOL expression. For example, the function that adds 5 to a number can be written $\lambda x.(x+5)$. This is a function, so we can apply it to a value and get a result: to apply it to, e.g., 3, we write $(\lambda x.(X+5))(3)$, and the result is, of course, 8. Function application is called β-reduction in the lingo of the λ-calculus formalism. Church must have been fond of Greek letters since he defined another operation: α-conversion, which is simply the possibility to change a variable's name once a λ operator binds it: $\lambda x.(x+5)$, $\lambda y.(y+5)$ or $\lambda\heartsuit.(\heartsuit+5)$ denote precisely the same function—they are easily obtained from each other through α-reduction.

In our case, we will write $\lambda X.\mathsf{sleeps}(X)$ as the function that assigns to any value of X the predicate sleeps(X).

The next question we must ask is: what will be σ_1, the converse of the syntactic rule s_1? The answer is simple: it will be β-reduction (alas function application). We will apply the function λx.sleeps(x) to the constant Clara to obtain sleeps(Clara). Here is the "semantic tree" we get:



and the converse σ_1 of the syntactic rule $s_1\colon$ S $\rightarrow$ NP VP will be

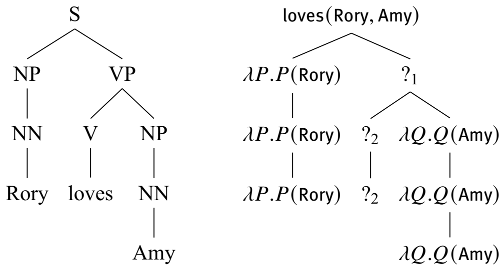
$$v(\mathrm{S}) := (v(\mathrm{VP}))(v(\mathrm{NP})),$$

where the parentheses are used for different reasons: the parentheses surrounding v(VP) indicate that it is a function, and the parentheses surrounding v(NP) indicate that it is the argument of this function.

☞ 95 One may wonder: since English is an SVO language, in the right part of s_1 we have NP first and then VP. What happens in languages where the order is reversed? In that case, we can either define σ_1 the other way around or perform an operation ☞ 114 called *type raising*: instead of using a FOL constant to represent "Clara," we use the following expression: $\lambda P.P(\mathsf{Clara})$. Intuitively, this means: "Give me a predicate and I will apply it to Clara," i.e., "ask me any question about Clara and I will reply by yes or by no." It is easy to verify that this expression does the job:

$$(\lambda P.P(\mathsf{Clara}))(\lambda x.\mathsf{sleeps}(x)) = (\lambda x.\mathsf{sleeps}(x))(\mathsf{Clara}) = \mathsf{sleeps}(\mathsf{Clara}).$$

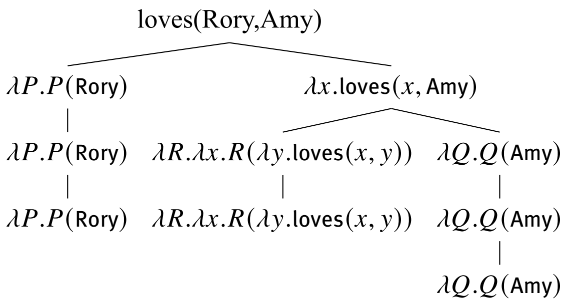
Now, we analyze a sentence with a binary predicate: "Rory loves Amy." The FOL formula we are looking for is $\mathsf{loves}(\mathsf{Rory}, \mathsf{Amy})$. To keep some symmetry between Rory and Amy, we can decide that we will represent both names by typed-raised expressions $\lambda P.P(\mathsf{Rory})$ and $\lambda P.P(\mathsf{Amy})$. But what FOL expression will represent "loves"? Here is what we got for the moment:



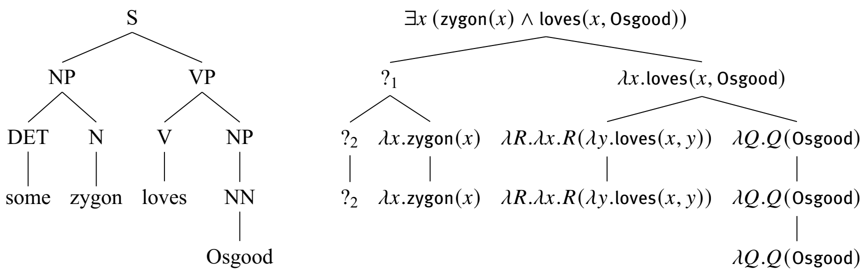
Node $?_1$ is simple to guess: we want a λ-prefixed predicate such that when we replace its bound variable with "Rory" we get $\mathsf{loves}(\mathsf{Rory}, \mathsf{Amy})$. This can only be $\lambda x.\mathsf{loves}(x, \mathsf{Amy})$. But what about $?_2$? That's a tricky one. We know that the expression we seek must start with a λ-prefixed predicate variable (since we are going to apply it to $\lambda Q.Q(\mathsf{Amy})$), and we know that it must contain loves(x,y) as well as λx and λy. This solution may be hard to find it, but it is easy to check that it is correct:

$$\begin{aligned}(\lambda R.\lambda x.R(\lambda y.\mathsf{loves}(x, y)))(\lambda Q.Q(\mathsf{Amy})) &= \lambda x.(\lambda Q.Q(\mathsf{Amy})(\lambda y.\mathsf{loves}(x, y)))\\ &= \lambda x.((\lambda y.\mathsf{loves}(x, y))(\mathsf{Amy}))\\ &= \lambda x.\mathsf{loves}(x, \mathsf{Amy}),\end{aligned}$$

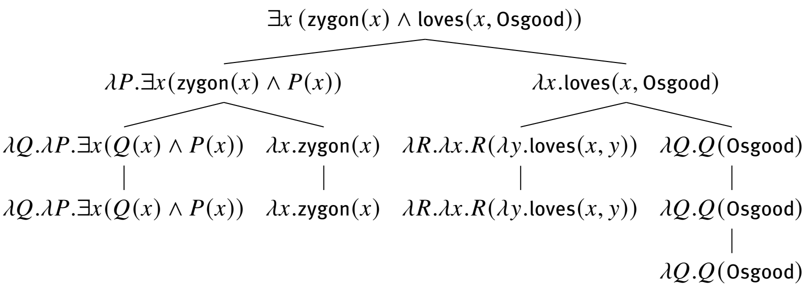
as requested. Here is how our semantic tree looks like now:



How about a sentence with an indefinite article? Let us take "Some zygon loves Osgood". A reasonable way of representing this sentence in FOL would be to cut it into two propositions: "There exists an x that is a zygon" and "x loves Osgood." The resulting FOL formula would be as follows



where we have represented the word "zygon" by $\lambda x.\mathsf{zygon}(x)$ since it is a property of an individual. Once again, $?_1$ is trivial: it can only be $\lambda P.\exists x(\mathsf{zygon}(x) \wedge P(x))$, since this will allow us to replace P by the loving-Osgood-predicate. To find $?_2$ we add just anotherλ-bound predicate variable R to $?_1$:



This example illustrates a frightening fact: Grammatical words tend to have very complex representations in Montague grammar. In her novel *The Semantics of Murder*, Campbell [8] describes the discovery of formal semantics by students of linguistics in the sixties (in this extract, "Robert" is the fictional alter ego of Montague and "Jay," his brother, sitting among the audience in the lecture room):

> 'Let's take a simple sentence like "Every man talks". Now the usual translation in logic would be

$$\forall x\, \mathsf{man}(x) \to \mathsf{talks}(x).$$

'But this is clearly unsatisfactory.' The class regarded his formula in cowed silence. A girl seated in front of Jay turned to her classmate and shrugged with a nervous grin, pointing to a large question mark with a bubble dot on her open notebook.

'So this is the solution I'm going to propose,' continued Robert.

$$\big(\lambda Q.\big((\lambda P.\forall x\, (Q(x) \to P(x)))\big)(\lambda y.\mathsf{talks}(y))\big)(\lambda z.\mathsf{man}(z)).$$

Jay watched the students' bobbing heads, mouthing as they transcribed, diligently counting the brackets. [Robert writes the semantic tree of the sentence] The chalk broke on the board and clattered to the floor at Robert's feet. The girl in front of Jay shook her head. Robert tapped a finger on the lectern and Jay could sense his brother's waning enthusiasm. His gaze traveled over the audience in barely concealed frustration. 'My God, this is even worse than I thought. We are all wasting our time,' and he grabbed his jacket and strode from the theatre. [8, pp. 57–58]

What the fictional Montague is doing here is just writing a general λ-expression that can be applied to any pair of predicates and then applying it to the specific predicates $\lambda y.\mathsf{talks}(y)$ and $\lambda z.\mathsf{man}(z)$. The students' frustration is understandable since, in the sixties, linguistics students were not accustomed to heavy logical formalism. However, the reference to murder in the novel's title is not due to the frustration of Montague's students. Montague was murdered under obscure circumstances. After his death, people discovered he lived parallel lives as a University professor, parish organist, real estate agent, and gay night club attendant. Astonishingly, no Hollywood blockbuster has yet been dedicated to Montague's life (and death).

6.5.2.1 Python Implementation of Formal Semantics

The *nltk* Python module provides a `LogicParser` class, with a `parse` method, that allows parsing FOL and λ-calculus expressions and referring to them by variables. A `ApplicationExpression` method will apply one expression to another, and a `simplify` method will perform all possible β-reductions. Let us take the "Rory loves Amy" example and rewrite it in Python:

```
import nltk
lp = nltk.sem.logic.LogicParser()
np2 = lp.parse(r'\Q.Q(Amy)')
v = lp.parse(r'\R.\x.R(\y.loves(x,y))')
vp = nltk.ApplicationExpression(v,np2)
print(vp.simplify())
```

(The backslash represents the λ symbol.) The result of this code is `\x.loves(x,Amy)`, which is exactly the expression of the VP node in our tree on p. 171. We can continue:

```
np1 = lp.parse(r'\P.P(Rory)')
s = nltk.ApplicationExpression(np1,vp)
print(s.simplify())
```

and the result will be `loves(Rory,Amy)`, which means that we have successfully obtained the semantics of the sentence "Rory loves Amy" out of the expressions of the leaves of the tree by using the compositionality principle and by applying FOL application on every intermediate node.

The exciting feature of *nltk* is that you can use FOL and -expressions as values of a feature in a feature-based grammar. By starting with the FOL and λ-calculus expressions corresponding to lexical items and then moving up using application of expressions, one arrives at the FOL expression of the entire sentence. ☞ 265

Here is the feature-based grammar for the "Rory loves Amy" example:

```
% start S
S[SEM=<?np(?vp)>] -> NP[SEM=?np] VP[SEM=?vp]
VP[SEM=<?v(?np)>] -> V[SEM=?v] NP[SEM=?np]
V[SEM=<\R.\x.R(\y.loves(x,y))>] -> 'loves'
NP[SEM=<\Q.Q(Amy)>] -> 'Amy'
NP[SEM=<\Q.Q(Rory)>] -> 'Rory'
```

which is just the usual grammar S → NP VP, VP → V NP, etc., to which we added the `SEM` feature, the values of which are FOL expressions as long as we are just as a step above word forms, and application of feature variables, afterward. Here is the result:

```
(S[SEM=<loves(Rory,Amy)>]
  (NP[SEM=<\Q.Q(Rory)>] Rory)
  (VP[SEM=<\x.loves(x,Amy)>]
    (V[SEM=<\R x.R(\y.loves(x,y))>] loves)
    (NP[SEM=<\Q.Q(Amy)>] Amy)))
```

which is exactly the tree of p. 171.

6.5.2.2 Formal Semantics through CCGs

Montague used Chomsky's rewrite rules and constituent syntax trees, but formal semantics can very well be based on other syntax formalisms. Let us see how the same sentences can be analyzed using Steedman's combinatory categorial grammars. ☞ 109

First is how CCGs deal with SVO and OVS orders. We have seen that in Montague's approach, type-raising solves the problem of word groups being in the wrong order. This is not necessary for CCGs since the category of a verb already contains information on the subject's position and potential object. Steedman defines *forward* and *backward* functional applications as follows [87, p. 37]: ☞ 95

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : f\,a \qquad (>)$$
$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f\,a \qquad (<)$$

What is meant in the first line is that, on the syntactic level, whenever we have a constituent of category X/Y followed by a constituent of category Y we can derive a constituent of category X. Also, if the semantics of X/Y is f and the semantics of Y is a, then the semantics of the resulting constituent is $f\,a$, that is, the functional application of f on a. On the second line, we have the inverse situation: Y precedes $X\backslash Y$, but the semantics of the resulting constituent is the same. As Steedman says,

> It is important to notice that these two rules apply an identical compositional-semantic operation—functional application—in both syntax and semantics. [87, p. 37]

This being said, here is the CCG diagram of "Clara sleeps," together with the semantics of the sentence in first-order logic:

$$\begin{array}{cc}
\textsf{Clara} & \textsf{sleeps} \\
\mathrm{NP}\colon \textsf{Clara} & \mathrm{S\backslash NP}\colon \lambda x.\textsf{sleeps}(x) \\
\hline
\multicolumn{2}{c}{\mathrm{S}\colon \textsf{sleeps}(\textsf{Clara})}
\end{array}{<}$$

A single backward application derivation is sufficient to obtain the desired FOL formula. And here is our second example:

$$\begin{array}{ccc}
\textsf{Rory} & \textsf{loves} & \textsf{Amy} \\
\mathrm{NP}\colon \textsf{Rory} & \mathrm{(S\backslash NP)/NP}\colon \lambda y.\lambda x.\textsf{loves}(x,y) & \mathrm{NP}\colon \textsf{Amy} \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathrm{S\backslash NP}\colon \lambda x.\textsf{loves}(x,\textsf{Amy})\quad >} \\
\hline
\multicolumn{3}{c}{\mathrm{S}\colon \textsf{loves}(\textsf{Rory},\textsf{Amy})\quad <}
\end{array}$$

In this case, a forward application followed by a backward application was sufficient to provide the desired result. As for the "Some zygon loves Osgood" sentence, this time we will indeed use type raising, but it is incorporated in the lexical category of the quantifier "some" [87, p. 71]:

$$\begin{array}{cccc}
\text{some} & \text{zygon} & \text{loves} & \text{Osgood} \\
\mathrm{(T/(T\backslash NP))/N}\colon \lambda P.\lambda Q.\exists x\,(P(x)\wedge Q(x)) & \mathrm{N}\colon \lambda x.\text{zygon}(x) & \mathrm{(S\backslash NP)/NP}\colon \lambda y.\lambda x.\text{loves}(x,y) & \mathrm{NP}\colon \text{Osgood} \\
\cline{1-2}\cline{3-4}
\multicolumn{2}{c}{\mathrm{T/(T\backslash NP)}\colon \lambda Q.\exists x\,(\text{zygon}(x)\wedge Q(x))\quad >} & \multicolumn{2}{c}{\mathrm{S\backslash NP}\colon \lambda x.\text{loves}(x,\text{Osgood})\quad >} \\
\hline
\multicolumn{4}{c}{\mathrm{S}\colon \exists x\,(\text{zygon}(x)\wedge \text{loves}(x,\text{Osgood}))\quad >}
\end{array}$$

The notation "T" needs some explanation. When we type-raise a noun, it becomes (T\(T/NP))/N for "some parametrically licensed category for the language" [87, p. 57], which means that T is a placeholder for various categories. Therefore, when deriving line 3, we consider T as being S and get the desired result. It should be noted that in CCG, there is a type-raising combinator **T** (the *thrush*), which is used to type-raise nouns, but in our case type raising is already included in the category of the quantifier "some."

6.6 Discourse Semantics

Chomsky has considered the sentence the largest text unit, but a sentence does not live in isolation. There are relations between sentences, and to convince oneself, one only needs to take a paragraph of text[8] and randomly change the order of sentences: the paragraph will most probably become incoherent. Research in discourse analysis has mainly followed two directions: the first one is to study the relations between

[8] Other than *exquisite corpses*, see p. 404.

sentences (or clauses), the most important approach being *Rhetorical Structure Theory* (RST); the other direction consists in representing the semantics of a sequence of sentences as an accumulation of first-order logic expressions, and thereby expand Montagovian semantics to multiple sentences, the most important achievement being *Discourse Representation Theory* (DRT).

6.6.1 Rhetorical Structure Theory

Rhetorical Structure Theory, introduced by Mann & Thompson [57] and subsequently formalized by Marcu [58], breaks a text into non-overlapping spans (prosodic units, turns of talk, sentences, etc.) and then studies discourse relations between these units.

Most of these relations are binary and asymmetric: in this case, we have a *nucleus* and a *satellite*. These relations are called *hypotactic* (the Greek term for subordination); some relations are *n*-ary and symmetric: we have two or more nuclei of equal importance, these relations are called *paratactic*.

Here is a list of hypotactic relations, taken from [57, 86]. Mann & Thompson [57] emphasize that this list is open and is "susceptible to extension and modification for particular genres and cultural styles." In describing each relation, we use the following symbols: R stands for the reader, W for the writer, N for the nucleus, and S for the satellite.

CIRCUMSTANCE	R recognizes that the situation presented in S provides the framework for interpreting N
SOLUTIONHOOD	the situation presented in N is a solution to the problem stated in S
ELABORATION	S presents additional details about the situation, which is presented in N
BACKGROUND	S increases the ability of R to comprehend an element in N
ENABLEMENT	R comprehending S increases R's potential ability to perform the actions presented in N
MOTIVATION	as in ENABLEMENT, but replace "ability" by "desire"
EVIDENCE	An evidence satellite is intended to increase the reader's belief in the nuclear material
JUSTIFICATION	A justificatory satellite is intended to increase the reader's readiness to accept the writer's right to present the nuclear material
VOLITIONAL CAUSE	S presents a situation that could have caused the agent of the volitional action in N to perform that action
NON-VOLITIONAL CAUSE	S presents a situation that, by means other than motivating a volitional action, caused the situation presented in N
VOLITIONAL RESULT	S presents a volitional action or a situation that could have arisen from a volitional action and N presents a situation that could have caused the situation presented in S
NON-VOLITIONAL RESULT	S presents a situation that is not a volitional action, and N presents a situation that caused the situation presented in S
PURPOSE	S presents a situation to be realized through the activity in N
MEANS	S provides information that makes the realization/execution of N more probable or simple (e.g., an instrument)

ANTITHESIS	the situations presented in N and S are in contrast. Because of their incompatibility, R's positive regard for the situation presented in N is increased ("but")
CONCESSION	W acknowledges a potential incompatibility between the situations presented in N and S, but W regards the situations presented in N and S as compatible, and this compatibility increases R's positive regard for the situation presented in N ("although")
CONDITION	realization of the situation presented in N depends on the realization of that presented in S
OTHERWISE	realization of the situation presented in N prevents the realization of the situation presented in S
UNLESS	N is only realized if S is not being realized
INTERPRETATION	S relates the situation presented in N to a framework of ideas not involved in N itself
EVALUATION	S relates the situation in N to the degree of W's positive regard toward the situation presented in N
RESTATEMENT	S restates N, where S and N are of comparable bulk
SUMMARY	S presents a restatement of the content of N, which is shorter in bulk

And here is a (much shorter) list of paratactic relations, including the catch-all relation JOINT, which we use when we consider that there is no discourse relation between two or more units of when we have no clue on the author's intentions.

SEQUENCE	a succession relationship between the situations presented in the nuclei
LIST	the nuclei provide information that can be recognized as related, enumerating. They all contribute to the text function in the same way
CONJUNCTION	like LIST but linked by coordinating conjunctions
CONTRAST	two nuclei that are comprehended as the same in many respects but differ in a few respects and are compared for one or more of these differences
JOINT	the nuclei provide different kinds of information, which are not of the same type; yet they are not in an identifiable semantic or pragmatic relation, nor do they form an enumeration. Still, there is a coherent link, as they contribute in similar ways to the overall text function

These relations, whether paratactic or hypotactic, are applied to individual text units and blocks resulting from the accumulation of text units. This is best illustrated by an example taken from Marcu [58, pp. 29–31], quoting Webber [97, p. 115]:

> [There are two houses you might be interested in:]$_A$ [House A is in Palo Alto.]$_B$ [It's got 3 bedrooms and 2 baths,]$_C$ [and was built in 1950.]$_D$ [It's on a quarter acre, with a lovely garden,]$_E$ [and the owner is asking $425K.]$_F$ [But that's all I know about it.]$_G$ [House B is In Portola Valley.]$_H$ [It's got 3 bedrooms, 4 baths, and a kidney-shaped pool]$_I$ [and was also built in 1950]$_J$ [It's on 4 acres of a steep wooded slope, with a view of the mountains.]$_K$ [The owner is asking $600K.]$_L$ [I heard all this from a friend,]$_M$ [who saw the house yesterday.]$_N$ [Is that enough information for you to decide which to look at?]$_P$

We have indexed text units by letters between A and P. The RST diagram of this paragraph can be seen in Fig. 6.2. Units B–F and H–L form unit LISTS, so we can consider blocks B–F and H–L. The former is involved in an ANTITHESIS relation with G (typically ANTITHESIS relations start with a "but": here it is "but that's all I know about it"). Between M and N, there is an ELABORATION relation ("[a friend]

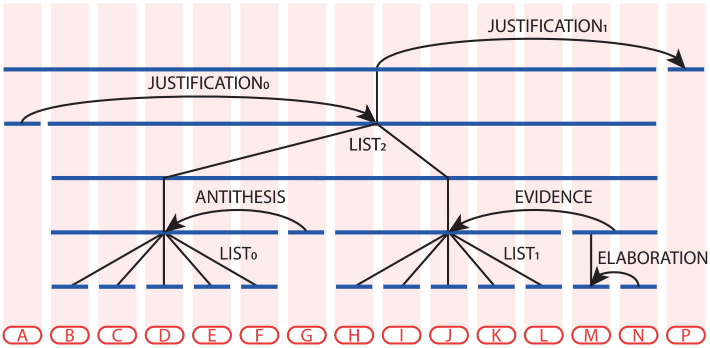


Fig. 6.2 RST diagram of the "Palo Alto houses" paragraph, adapted from marcu [58, p. 30]

who saw the house yesterday" elaborates on "I heard all this from a friend"), so that we can consider them as a block M–N. Between this block and the H–L block, there is an EVIDENCE relation: information about the Portola Valley house is evidenced by the testimony of the speaker's friend. We have two contiguous blocks at this level, B–G and H–N. We consider that they form a LIST. This list is justified by A so that we get a new block A–N, and this "super-block" justifies the final unit P ("Is that enough information for you to decide which to look at?"), and we are done.

Detection of discourse relations is relatively straightforward in the presence of explicit markers (e.g., the presence of "because" is a strong hint for a CAUSE relation while "afterward" rather leads us towards a SUCCESSION relation). In the absence of such markers, the problem is intrinsically difficult (the choice of discourse relation can be subjective since it depends on the interpretation of the text). A useful approach to detect discourse relations is to use *tests*. For example, the *insertion test* [79, p. 142] consists of adding explicit markers and choosing the one that provides a coherent discourse of similar interpretation as the original discourse. If we take the two units

(1) the Doctor visited Rory and Amy for Christmas
(2) they (= Rory and Amy) were very happy

a possible interpretation is "(2) because (1)" (CAUSE). An interpretation "(1) but (2)" would be incoherent, so that ANTITHESIS is excluded. Also, "(1) during a period where (2)" is possible, meaning that (1) and (2) are independent events/situations that happened at the same time, the relation would be merely a JOINT. After excluding markers that produce an incoherent discourse, knowledge of the context is necessary to evaluate the pertinence of the remaining candidate discourse relations.

6.6.2 Discourse Representation Theory

Discourse Representation Theory (DRT) will gather agents (called "referents") and formulas (called "conditions") from multiple sentences and build a structure containing both new symbols for each sentence and identities between symbols with the same reference. For example, the discourse

> Amy bought a pie.
> Rory ate it.

will be represented in DRT as

$x\ y\ z\ u\ e\ e'\ t\ t'$
Amy(x)
pie(y)
e:bought(x,y)
$t < n, e \subseteq t$
Rory(z)
e':ate(z,u)
u=y
$t' < n, e' \subseteq t', t' > t$

where e and e' are events, t, and t' time spans, n stands for "now," $e \subseteq t$ means that event e took place during time span t, and > is temporal order.

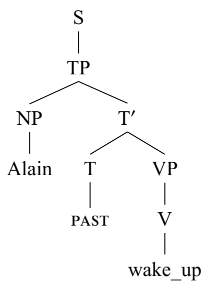
Such a box is called a DRS (*discourse representation structure*). In the upper part of the box, we place all referents, and in the lower part, a conjunction of logical formulas, including identities between co-referential referents. This may seem like an extra burden. Why not keep only one of the two symbols? After all, they are equal. It allows for keeping the paragraph structure: every sentence uses its symbols, and identities between them explicitly indicate co-reference. According to Kamp, the rationale of this approach is based on cognitive modeling:

> From the very beginning, one of the strong motivations of DRT was the desire to capture certain features of how interpretations of sentences, texts, and discourses are represented in the interpreter's mind, including features that cannot be recaptured from the truth conditions that the chosen interpretation determines. [43]

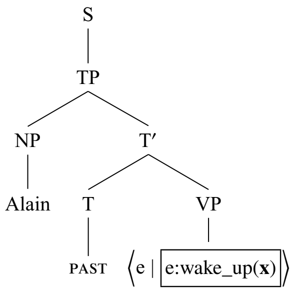
Let us see how DRSs are built through a simple example, taken from Kamp [43]:

> Alain woke up. His wife was smiling.

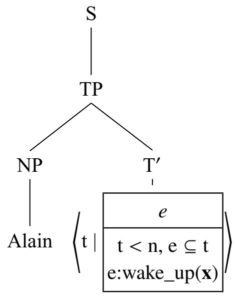
The syntax tree of the first sentence is written as



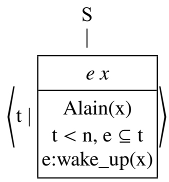
☞ 100 where, as per Chomsky's transformational grammar theory, we have a tense phrase (TP) in which we separate information about tense (PAST) from the lemma of the verb ("wake_up"). We will now write pairs ⟨ unbound referents | DRS ⟩, where unbound referents will be bounded progressively. We start with the verb node:



where **x** is a placeholder filled by the referent of the verb's subject once we arrive there. The event e we introduce is still unbound. It will be bound with time information: saying that event e occurred during time span t binds the former to the latter so that e now is a referent of our DRS:



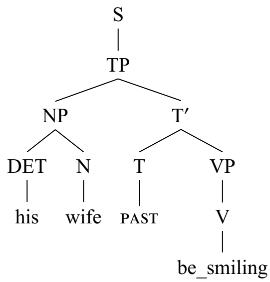
The next step takes the NP branch into account and fills the placeholder **x** with a new referent x (the symbol is not bold), that represents Alain:



Now we have to bind t: as this sentence is the first one of our discourse, we assume that there is no previous temporal reference and that we can assume an existential quantifier ∃ t. This means that t becomes a referent of our DRS:

$$\begin{array}{|c|}\hline \textit{t e x} \\ \hline \text{Alain(x)} \\ \text{t} < \text{n, e} \subseteq \text{t} \\ \text{e:wake\_up(x)} \\ \hline \end{array}$$

Let us see how the second sentence gets incorporated into the DRS of the first. We have the syntax tree (which is quite similar to the first one):



Repeating the same steps, we arrive at

$$\left\langle t', u \mid \begin{array}{|c|}\hline s\ x' \\ \hline \text{wife}(x',u) \\ t' < n,\ s \subseteq t' \\ s{:}\text{be\_smiling}(x') \\ \hline \end{array} \right\rangle$$

where we use s as a referent of be_smiling since we consider it rather as a situation than an event.

The DRS of the first sentence functions as *discourse context* for the interpretation of the DRS of the second. By matching u's masculine gender (the example originated from the sixties when same-sex marriages were not yet possible) and the masculine gender of the name "Alain" (background knowledge), we infer that u is co-referential with the referent x of Alain(x). Also, the discourse relation (in the sense of RST) between the two sentences (SUCCESSION) allows us to guess that the two events/situations e and s took place during the same time span. So t′, and u become referents, and we get the DRS

$$\begin{array}{|c|}\hline t\ e\ x\ t'\ s\ x'\ u \\ \hline \text{Alain}(x) \\ t < n,\ e \subseteq t \\ e{:}\text{wake\_up}(x) \\ \text{wife}(x',u) \\ t' < n,\ s \subseteq t' \\ t' = t,\ u = x \\ s{:}\text{be\_smiling}(x') \\ \hline \end{array}$$

This example only scratches the surface of DRT. We have only briefly mentioned that to conclude that two referents are co-referential, we must look for background knowledge. In earlier versions of DRT [42], the method of DRS construction involved temporary DRSs called *presuppositions*. In our example, this would generate the conditions male(u) and human(u) in the second sentence because of the gender of the pronoun "His" and the fact that to have a wife, one must be human (at least up to now). These presuppositions were used for the unification of DRSs. Another phenomenon of implicit information is *implicatures*.

6.7 Implicatures and Conversation Maxims

In the seventies, the philosopher Grice introduced the term *conversational implicature* for information that is not explicit in conversations or other kinds of text. Often, we use sentences that could be straightforwardly rewritten as logical formulas, but the meaning we intend includes additional, implicit information. Take, for example, the sentence, "Some students failed." One may interpret "some" as "at least one," in which case the formula would be

$$\exists X \ \mathsf{student}(X) \wedge \mathsf{failed}(X),$$

or one may also interpret it as "at least two," in which case the formula would be

$$\exists X, Y \ \mathsf{student}(X) \wedge \mathsf{failed}(X) \wedge \mathsf{student}(Y) \wedge \mathsf{failed}(Y) \wedge X \neq Y$$

but how about $\forall X \ \mathsf{student}(X) \wedge \mathsf{failed}(X)$?

This implies the formula $\exists X \ \mathsf{student}(X) \wedge \mathsf{failed}(X)$ and therefore, logically, if it is true that all students failed, then it is also true that some students failed. But this is not our intention: when we say "some students failed," we mean "some, *but not all*, students failed," otherwise, we would have said, "*all* students failed."

Grice called the implicit fact that "not all students failed" a *conversational implicature*. He uses the verb *to implicate* instead of the verb *to imply* and, as underlined by Bach [2, p. 22], there is a reason for it: when a sentence is true, then whatever it implies (in the sense of FOL) must be true as well, while an implicature can be false. Implication is based on *semantic content* while implicature is based on *communicative intention*. The same sentence, "the train leaves in five minutes," can implicate "we still have five minutes" if the speaker and listener are inside the train; "let us hurry" if they are in front of the railway station; "we missed it" if they are blocked in a traffic jam on the other side of the town.

Implicatures are not arbitrary but are part of a framework that Grice called *maxims*. Maxims contribute to the efficiency of conversation. They are cooperative efforts between communicating agents to optimize the information exchanged. They are presumptions that listeners rely on, and speakers exploit. Grice provided the following list of maxims, which we illustrate by imagining "wrong" answers to the question, "How do I get to the Empire State Building?":

1. *quantity*: make your contribution as informative as is required (don't reply by "take the plane to JFK Airport"),
2. do not make your contribution more informative than is required (it is needless to give the number of steps of the staircase of the New York Penn Station),
3. *quality*: do not say what you believe to be false (don't say "you can ride a unicorn to the Empire State Building"),
4. do not say that for which you lack adequate evidence (don't say "beware of the hurricane" if you are not sure there will be one),
5. *relation*: be relevant (don't describe the rhinoceros' sex life when asked how to go to the Empire State Building),

6. *manner*: avoid obscurity of expression (don't use words such as "to flabbergast" or "limerence," even if their meaning is relevant),
7. avoid *ambiguity* (don't say "take any means of transportation and eventually you will arrive there"),
8. be *brief* (don't spend 15 minutes answering when you can give a complete answer in 30 seconds),
9. be *orderly* (give directions in the correct order. An answer such as "Take the train from JFK to Penn Station. Take the plane from Paris to JFK. Walk from the Penn Station to the Building" is incoherent because the steps are not in the right order).

What about our example of "some students" implicating "not all students"? If the reason we said "some" and not "all" is because we are unsure (we don't know whether there is at least one student that succeeded), then we use Maxim 4: we do not have enough evidence to say that all students have failed, so we speak of "some students." If we know that a group of students succeeded, saying "some, but not all" would emphasize that not all failed—we consider this information unnecessary, so according to Maxim 2, we do not provide it.

Interestingly, as Grice [31] observes, besides being fulfilled, maxims can also be *violated*, *opted out*, *in a clash situation* or *flouted*.

When we pretend we know a fact and give an answer despite our superficial knowledge of it, we violate Maxim 4. *Violation* is quiet and unostentatious: as Grice says, we are liable to mislead in such a situation. Depending on the gravity of the situation, such a violation can be negligible or have grave consequences, as when somebody suggests injecting disinfectant as a treatment for COVID-19.

Opting out explicitly states that we do not fulfill a given maxim, as when we say, "I cannot say more, my lips are sealed," and thereby violate Maxim 1.

Clashes occur when maxims happen to be contradictory: I may not give sufficient information (therefore not fulfilling Maxim 1) to respect Maxim 3 (and hence avoid giving uncertain information). It is our responsibility to rank the importance of maxims (and the consequences of violating them) in every situation and find optimal ways to handle maxim clashes.

From a linguistic point of view, the most interesting case is the one of maxim *flouting*, i.e., blatantly failing to fulfill one or more maxims. Grice gives an amusing example of flouting Maxim 1: a professor writes a recommendation letter for a student for a position in the Philosophy Department, as follows:

> Dear Sir, Mr. X's command of English is excellent, and his attendance at tutorials has been regular. Yours sincerely, etc. [31, p. 33]

It is immediately apparent that this letter does not mention the candidate's competencies of activities in philosophy. It obviously and very strongly violates Maxim 1. The only reason this may happen is that the professor is reluctant to write down her honest opinion, implicating by the absence of this information that the candidate is not good.

Flouting of maxims is also used in *irony* (saying "What a beautiful day!" amid ☞ 162 a hailstorm), *metaphor* ("You are my sunshine"), *meiosis* (meaning "diminution" in

Greek, saying "we have a slight problem" when an intercontinental rocket reaches you in less than a minute) and *hyperbole* (meaning "exaggeration" in Greek, as in "we have the best fried chicken in the Universe"). In all these cases, it is Maxim 3 that is flouted: we deliberately say something false. Maxim flouting is also used in humor, as in the remarkable joke:

> A father's advice: "My son, two things are important in life. The first is that you should always say only half of what you know,"

flouting two things, the first of which is Maxim 1.

6.8 Psycholinguistic Aspects

6.8.1 Independence of Syntactic and Semantic Processing

Before we give a global overview of how the mind handles language, let us first return to the garden path sentences we saw in the chapter on syntax. A legitimate question would be: Do the semantics of these sentences interfere with their syntactic processing? In other words, do we perform simultaneous syntactic and semantic processing so that the latter can assist the former and make some garden-path sentences less garden-path-ish? ☞ 127

Ferreira & Crofton [17] describe the following experience: University of Massachusetts students (paid $5 an hour, which is reasonable, considering the fact that they had their head immobilized in front of a screen by a bite bar…) were exposed to sentences, and their eye movements tracked. The sentences were of four kinds, as in this example: ☞ 57

(a) The defendant examined by the lawyer turned out to be unreliable.

(b) The evidence examined by the lawyer turned out to be unreliable.

(c) The defendant that was examined by the lawyer turned out to be unreliable.

(d) The evidence that was examined by the lawyer turned out to be unreliable.

Sentences (a) and (b) are *reduced* (in the sense that they have no explicit marker of the relative clause) while (c) and (d) have the explicit marker "that was." In sentences (a) and (c), the subject is animate and, therefore, could be the agent that examines. In contrast, in sentences (b) and (d), the subject is inanimate and, therefore, cannot examine anything. The garden path sequence effect is that by the Minimal Attachment principle (p. 128), we first consider "examined" as being a past tense verb and not a participle, and it is the preposition "by" that forces us to reconsider the sentence's syntax. ☞ 128

What Ferreira & Crofton tested was the reading times per character (in ms) and per *zone*. Zones were defined as follows:

1. the first NP (zone $c - 2$),
2. the "that was" phrase if it existed.

3. the first verb ($c - 1$),
4. the *by* phrase or prepositional phrase (c),
5. the main verb and any auxiliaries ($c + 1$), and
6. the rest of the sentence, if the sentence continued past $c + 1$ ($c + 2$).

Only regions $c - 1$, c, and $c + 1$ were included in the data analyses.

Here are the results:

Condition	First pass $c-1$	First pass c	First pass $c+1$	Second pass $c-1$	Second pass c	Second pass $c+1$
Animate Reduced (a)	33.3	40.4	31.9	15.3	8.2	12.8
Animate Unreduced (c)	31.9	30.7	33.1	6.9	3.6	8.0
Inanimate Reduced (b)	37.7	38.4	32.6	12.6	14.9	16.9
Inanimate Unreduced (d)	30.1	30.3	28.6	0.0	0.0	2.5

What we can infer from these results is that (1) the presence of the explicit marker "that was" made the processing of the sentence easier and, in the case (d) even prevented a second pass, but, more importantly, (2) the fact that in the unreduced case, the first pass is *not* faster when the subject is inanimate, and the second pass is *longer* in the inanimate case. In other words, during the first pass, semantics do not interfere with syntax, and the sentence keeps its garden path effect even though "evidence" cannot examine anyone or anything. During the second pass, the semantic incongruity probably made the poor students stay longer on c and $c + 1$, trying to figure out what was (semantically) wrong.

The paper described three more experiments, and the general conclusion is that (we underline):

> At an empirical level, the experiments reported here provide evidence that some syntactic preferences, such as the preference for Minimal Attachment analyses, operate *largely independent* of semantic, pragmatic, and contextual factors. [17, p. 364]

6.8.2 Architecture of the Language Processing System

In this section, heavily inspired by Warren [96], we will consider how the various components of language production and comprehension systems fit together and interact.

Is language processing modular? There is some consensus in favor of an affirmative answer to this question, especially since the release of the book *The Modularity of Mind* by Fodor in 1983 [22]. Still, there is a lot of discussion about what we exactly mean by "modularity," i.e., what the modules are and how they interact.

In the previous section, we saw that syntactic and semantic processing are, in some sense, independent, and therefore one can assume that they belong to different modules. On the other hand, there are experiments exhibiting interferences between levels. One of them is the *Stroop effect*, one of the oldest psychological tests (1935) [88]. In the Stroop effect, participants are requested to name the color of a word written on the screen, and their reactions are monitored. At some point, the name of a color appears written in a different color (for example, the word “GREEN” written in red), which results in participants either giving the wrong answer or taking more time to answer. One could argue that this hints to the non-modularity of the mind since the visual perception of colors and the response to this perception are influenced by the semantics of the words read. A less-known fact about the Stroop effect is that it almost disappears when the participant’s response is given by mechanical and not by verbal means. This means that when the task is to “push on the red button when a word is red,” then subjects manage to perform it correctly even if the word says “GREEN,” and that the problems start when one has to use language production mechanisms to utter the word “red” while confronted with the printed word “GREEN.” Magen [56] argues that the Stroop effect is caused by several modules in the visual system, all endowed with perception and response selection processes. Also, different modules have different representations of colors (including verbal representations), and problems arise during translation between these representations.

An important argument favoring modularity is that two separate zones in the brain seem to be specialized, one in syntax and the other in semantics: the *Broca zone* and the *Wernicke zone*. People with brain damage in the Broca zone (*Borca aphasics*) typically have difficulty ordering words, but the words they will use will be meaningful. People with brain damage in the Wernicke zone (*Wernicke aphasics*) will speak fluently and syntactically flawlessly but without any discursive coherence or reference to the context. This description is a bit too crisp. In reality, the boundaries between language pathologies are somewhat fuzzy. Sometimes semantics may assist Broca aphasics in recovering syntactic information, for example, in *non-reversible* sentences such as “the mouse ate the cheese,” where “cheese” can hardly be the eater and therefore must necessarily be the eaten.

The common consensus is that modules exist but, according to Warren,

> are permeable and capable of sharing information during processing. [96, p. 222]

Harley [35] summarizes the overall structure of the language system in the diagram of Fig. 6.3.

The central element in this diagram is the “Verbal semantic system,” a place “where word meanings are stored and that interfaces with the other cognitive processes” [34, p. 464]. Above it, we see the “Visual semantic system,” where meaning is “indexed” by images. These two communicate in both directions.

Otherwise, the diagram is oriented top-bottom, from perceptory input (speech, writing) to physical output (spoken or written). “Auditory phonological analysis” will consider spoken input regarding the current language’s phonology. If a word is recognized as conformant to the phonological system, it is queried for in the “Phonological input lexicon.” If it is not found in that lexicon, it can still be lemmatized (“Lemmas”); otherwise, its meaning is sought in the “Verbal semantic system.” If a word does not match the language’s phonology, i.e., if it is a non-word, it can still be repeated without access to lexica. This is the route going through ‘Acoustic-to-phonological conversion,” in the sense that we will still use some phonological system to output the non-word. This is more-or-less what happens in the left column, the one of speech.

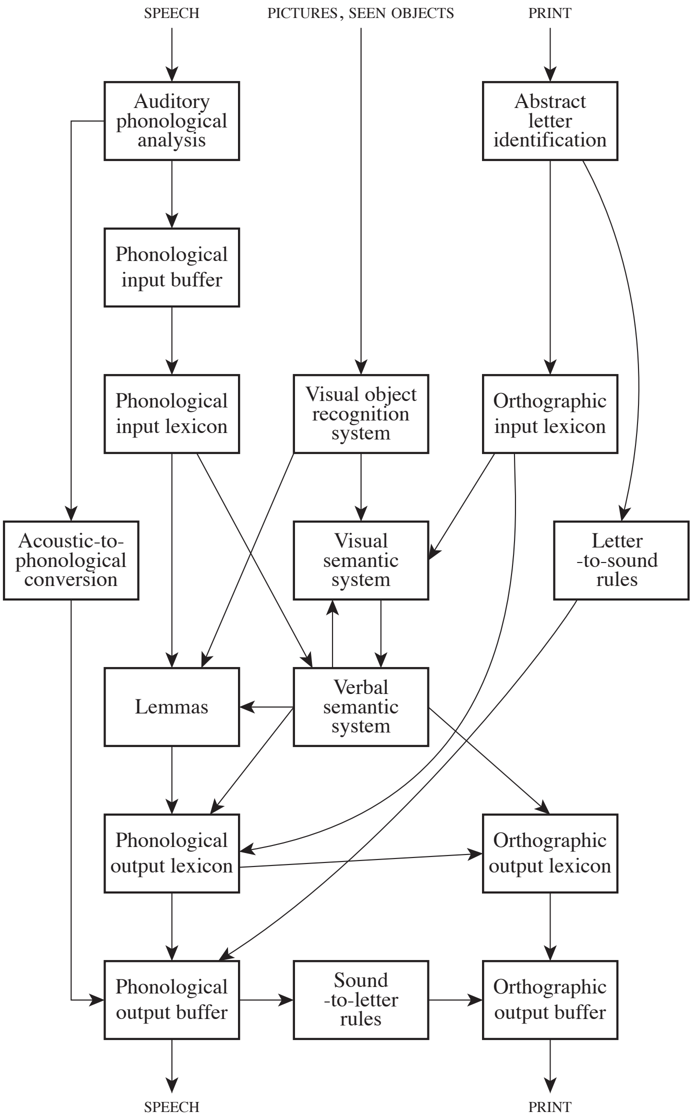


Fig. 6.3 The overall structure of the language system, taken from [34, p. 465]

When objects are seen, i.e., when pictures arrive in the brain, the "Visual object recognition system" attempts to recognize them. If they are recognized in the "Visual semantic system," they are verbalized and end up in the "Verbal semantic system." In some pathological cases, an object is named (and hence is matched with a lemma) but not recognized. This is why there is an edge from the "Visual object recognition system" to "Lemmas" without going through semantics.

The right column is the one for reading and writing. Harley [34] calls the first step “Abstract letter identification,” but this is only partly true because (1) the writing systems of the majority of people in the world do not use letters, so this is a very Eurocentric point-of-view, and (2) words are recognized as shapes and not by their individual γ-graphs. Otherwise, the right part of the diagram is quite similar to the ☞ 40 left part. When a word is not recognized in the “Orthographic input lexicon,” it is considered a non-word, and all we can do with it is apply “letter-to-sound“ rules (in fact, Harley means “grapheme-to-phoneme” rules) and pronounce it. If it is recognized as a (written) word, then we can either understand its meaning (going through the “Visual semantic system”) or fail to understand it. In this case, we go to the “Phonological output lexicon” to output it without understanding it.

Besides input and output, the diagram contains “buffers” in that some frequent data do not need to be looked up in the lexica each time they appear but are buffered instead.

Many of these abstract modules are conjectured to exist because of pathological cases. For example, there exist patients who are unable to name a cat by sight but can name it if they hear it mew or if they are given one to touch [34, p. 341], a condition called *optic aphasia* [5]. Psychologists argue that if this can happen, the channel between the visual semantic system and the verbal semantic system is broken. However, there is still a connection between a sound-based semantic system (recognizing entities by their sound) or a tactile semantic system (recognizing entities by touching them) and the verbal semantic system. This is a strong claim, fragilized by the fact that we use isolated and rare examples to draw general conclusions. The bottom line is that the diagram of Fig. 6.3 may change significantly in the future as evidence accumulates.

6.9 Further Reading

6.9.1 Literature

Riemer’s *Introducing Semantics* [73] is an excellent global textbook on semantics. For the reader seeking a comparative overview of semantic theories, Geeraerts’ *Theories of Lexical Semantics* [25] is a must, and a large part of this chapter is inspired by it.

Cann’s *Formal semantics* [9] is a solid classic for the reader interested in Formal Semantics. For readers of French, Roussarie’s *Sémantique formelle. Volume 1: Introduction à la grammaire de Montague* [78] is a modern alternative to [9], a very pleasant read whose second volume is eagerly awaited.

For the reader interested in Cognitive Semantics, we recommend the three volumes of Dąbrowska & Divjak’s *Cognitive Linguistics* series: *Foundations of Language*, *Key Topics*, and *A Survey of Linguistic Subfields* [50, 92, 52]. Geeraerts, mentioned in the previous paragraph, is also the co-editor, together with Cuyckens, of the very complete *Oxford Handbook of Cognitive Linguistics* [26], a volume of 1,365 pages!

Huang's *Pragmatics* [37] is a good introduction to Pragmatics. Also very interesting, and a source of inspiration for this chapter, is the very recent *Pragmatics in English* [83] by Scott. A complete survey of the current stand of pragmatics is *The Oxford Handbook of Pragmatics* [38], edited by Huang.

As for semiotics, a trendy entry point is the general public book *Mythologies* by Barthes, published in 1957 and translated into English in 1972 [3]. It is a semiotic exploration of 29 topics from everyday life, such as "wrestling," "Garbo's face," "the Citroën DS" (a mythical car with the particularity that the initialism DS pronounced "dé-éss" in French is homophonic to "déesse"$_{\text{FR}}$ ⌊goddess⌋), "striptease," "Einstein's brain," etc. To celebrate the 50 years of *Mythologies*, a multi-author "remake" called *Nouvelles Mythologies* [24] has been published in 2014. It covers the most heteroclitic topics of the 2010s. Other books by Barthes are hard to read because of his very specific style, which has been parodied in *Le Roland-Barthes sans peine* ⌊Roland Barthes made easy⌋ [7] published in 2017.

Chandler's *Semiotics, the basics* [10] (also available as an audiobook) is a fine introduction to semiotics, more than enough for the needs of Natural Language Processing. Equally exciting and accessible is Sebeok's *Signs: An Introduction to Semiotics* [84], which also deals with sign creation by living organisms. Nöth's *Handbook of Semiotics* [64] is a classic, with chapters on various very interesting subjects, including "Universal Language," "Body Language," "Gaze," and "Magic." A more modern version is Trifonas' *International Handbook of Semiotics* [93], a book of more than 1,300 pages.

For comic strip fans, there is a volume of the *Introducing Books* series on Semiotics [11], and one on Barthes [91].

It may seem paradoxical, but to close this section on semantic readings, we have chosen to talk about *asemic writing*. According to Schwenger,

> asemic writing is writing that does not attempt to communicate any message other than its own nature as writing. It is, in the words of Michael Jacobson, "a shadow, impression, and abstraction of conventional writing" [40]. This distinguishes it from nonsense or gobbledygook or whatever may be produced by those emblematic monkeys whaling away at typewriters. […] Faced with signs that encourage us to read them at the same time that they systematically frustrate our reading, we may find ourselves replicating what psychiatry calls asemia—a condition in which the patient is unable to use or understand communicative symbols. [82, pp. 1–2]

Asemic writing is a form of art in the twilight zone between poetry and graphics (as in works by Cy Tombly, Michael Jacobson, Marian Billenga, Rosaire Appel, or Nyman). It is sometimes prominently displayed in public spaces (as in works by Chris-
☞ 50 tine Kettaneh [44]). A particular case of asemic writing is Xu Bing's *A Book from the Sky*, a 604-pages book composed using 4,000 meaningless sinograms. These sinograms use standard sinographic components so that the reader can capture their semanticities and phoneticities, producing a random mixture of potential meanings and phonetic realizations.

6.9.2 Science Fiction

One of the foundations of science fiction is *cognitive estrangement*, so semantic and pragmatic features are its core business. Quoting Suvin,

> SF is, then, a literary genre whose necessary and sufficient conditions are the presence and interaction of estrangement and cognition, whose main formal device is an imaginative framework alternative to the author's empirical environment. [89, pp. 7–8]

Using an alternative imaginative framework implies playing with meaning on every possible level. To keep this section brief, we will choose five works dealing with different aspects of meaning and meaning in context.

In *Lost Memory* (1952) by British author Peter Phillips, the phonetic transcription of words instead of standard orthography hints at the absence of lexical meaning. The story is about a civilization of robots founded centuries earlier by a computer that taught the robots English but erased all words related to humans and human activities. When a human crashes with his starship on this planet, the robots communicating with him do not understand parts of his vocabulary. This has dramatic consequences for him as he ends up dying burnt alive. The lack of understanding of the robots is marked by non-standard spelling, as in this dialog between the human and a robot:

> "Does blud mean anything to you?"
> "No," Chirik replied simply.
> "Or deth?"
> "No."
> "Or wor?"
> "Quite meaningless."

In trying to extract the human, robots use heat to melt the metal door. His last outcry is

> "Eeee ahahmbeeeeing baked aliiive in an uvennn ahdeeerjeeesussunmuuutherrr!"

(as transcribed by the robots). Once they enter the room, they see only

> … a soft, charred, curiously shaped mass of something which lay just inside the opening.
> "Undoubtedly a kind of insulating material," Chur-chur explained.

In his novel *Stranger in a Strange Land*, Heinlein uses a neologism, the verb *to grok*, to signify the Martian's profound understanding of the meaning of things, which goes way beyond human understanding. He uses the verb 471 times in the novel so that the act of "grokking" becomes an essential ingredient. The Martian starts by failing to "grok" human customs and traditions:

> Since he had almost finished the encyclopedia, he had read articles on "Religion," "Christianity," "Islam," "Judaism," "Confucianism," "Buddhism," and related subjects. He had grokked none of this.

Then he develops his own theology of the Word, not very different from John's "In the beginning was the Word":

> “Word is ‘God.’” He added, “You grok.”
>
> “I must admit I don’t grok.”
>
> “You grok,” Smith repeated firmly. “I am explain. I did not have the word. You grok. Anne groks. I grok. The grasses under my feet grok in happy beauty. But I needed the word. The word is God.”

(where Smith is the Martian). Finally he starts a

> new religious movement, in which the first step for adherents is learning to “grok” the Martian way. [85, p. 25]

Eugene Zamiatin’s *We* (1924) describes a dystopian society in which the singular first person pronoun is abolished so that people can think only collectively. Samuel Delany, in his novel *Babel-17* (1966), goes a step further. Babel-17 is a language that merges “I” and “you.” Quoting Başoğlu,

> However, the more she learns about the language, the more her perception and thoughts are unconsciously altered due to the way the language works. Babel-17 is designed as a computer language, and “it ‘programs’ a self-contained schizoid personality into the mind of whoever learns it, reinforced by self-hypnosis. This ‘personality’ has the general desire to destroy the Alliance at any cost”. Furthermore, the weapon-language’s preclusion of the concepts of “I” and “You” makes its qualia ineffable, thereby disguising and hiding it within the unconscious of the speaker. As a result, the ‘kernel of Wong’s being’ is gradually transformed into a saboteur of the Alliance government, into a traitor, while nothing changes on the level of consciousness. [4]

where Wong is the main character of the novel, a linguist. This approach applies the ☞ 7 Sapir-Whorf hypothesis to the alienation of universal semantic primes and presents the effects of this alienation as a “language virus.”

Harrison & Minsky’s *The Turing Option* (1992) is a very interesting read. On the cover of the French translation, it is advertised as a “thriller halétant”$_{\mathrm{FR}}$ ⌊breathtaking thriller⌋—in fact, it is no more breathtaking than a *Kojak* or a *Derrick* episode… But it is fascinating nevertheless because (a) Minsky explains his vision of how to build an artificial intelligence using as many agents as there are interacting agents in the Freudian psychic apparatus; (b) the book is set in 2023–2024 so that we can compare Harrison & Minsky’s predictions with the status of current technology (the authors overestimate 2023’s technology when they predict that biometric identity verification will be based on instant DNA analysis—they underestimate it when they still mention videotapes and storage devices with capacities of no more than a few gigabyte capacities); (c) the AI described in the book sounds very much like an embodied ChatGPT. Not to mention the fact that reading a book of fiction written not by just any author, but by one of the founding fathers of artificial intelligence, gives the reader a strange feeling of being in the interstice between fiction and reality…

Science Fiction often attributes extraordinary powers to specific uses of language. In Binet’s “La septième fonction du langage”$_{\mathrm{FR}}$ ⌊The Seventh Function of Language⌋ ☞ 7 builds upon an additional, fictional function of language, the *performative function*, discovered by Roland Barthes. It is for this function that Barthes has been assassinated. It allows the speaker to convince the listener under all circumstances and, therefore, is sought by politicians.

Ted Chiang, in his short story *The Story of Your Life* (which has been adapted to the cinema by Denis Villeneuve in 2016, under the title *Arrival*), goes one step further in the application of the Sapir Whorf hypothesis. As the main character, linguist Louise Banks, learns the Heptapod B language of the creatures visiting Earth, her perception of the world changes because the language uses simultaneity of events instead of causality. Quoting Glazier & Beck,

> Through these intimate encounters, Dr. Banks' subjectivity is consistently reshuffled while new subjective capacities are elicited that include the retrieval of future events, the ability to talk in languages unknown to her former self, and an ability to shift more fluidly from one thought paradigm to another. [28, p. 279]

The message of the short story is that only if we knew the future would we be able to exercise our freedom of will since we would know the consequences of our actions.

6.10 Exercises

Exercise 5-1: Find faux amis words in French, German, Italian, Spanish, and English

Two words w_1 and w_2 are “faux amis”$_{\text{FR}}$ ⌊false friends⌋ when (a) they are homographs, (b) they belong to different languages L_1 and L_2, (c) they are semantically significantly different, and (d) w_2 does not alternatively have the meaning of w_1 in L_1 and vice-versa. For example, “chair” is a faux ami between English and French since it exists in both languages and has different meanings: “chair”$_{\text{FR}}$ means ⌊meat⌋ in French, and “chair” does not mean “meat” in English nor does it mean a sitting device or the position of professor in French (nevertheless there is “chaire”$_{\text{FR}}$ ⌊position of professor⌋ in French when we add an ‘e’ suffix). While conditions (a), (b), and (d) are (relatively) binary, the third condition is gradual and gives the amount of “false friendship.”

Find faux amis in the languages stated, weighted by their amount of false friendship.

Two challenges are involved in this exercise: (a) How do you evaluate the semantic relatedness of words in different languages? (b) Words have many senses, how do you pinpoint the specific meanings that are faux amis?

Exercise 5-2: FCA

In FCA, what is the minimal number of features necessary to define five concepts? Illustrate the case of (car, human, airplane, canary, horse).

Can you find the same number of features to define the concepts of (set, monoid, group, ring, field) in Algebra? If not, can you explain why?

Exercise 5-3: The semantics of Lojban

Explore the semantics of the following text in the Lojban language:

> ni'o seldau lo selmi'ecatra fa lo du'u na ka'e katna vimcu lo stedu se cau lo nu da poi xadni zo'u ka'e katna vimcu lo stedu da

which is the translation[9] by .xorxes. (Jorge Llambías) of a sentence of Lewis Carroll's *Alice's Adventures in Wonderland*:

> The executioner's argument was that you couldn't cut off a head unless there were a body to cut it off from.

adapted to our needs by Martin Bays. This exercise is a continuation of Exercise 4-3.

References

[1] "A priest, a rabbit, and a minister". Arnold Zwicky's Blog https://arnoldzwicky.org/2020/02/26/a-priest-a-rabbit-and-a-minister/. 2020.

[2] Kent Bach. "The top 10 misconceptions about implicature". In: *Drawing the Boundaries of Meaning: Neo-Gricean studies in pragmatics and semantics in honor of Laurence R. Horn*. Ed. by Betty J. Birner and Gregory Ward. Vol. 80. Studies in Language Companion Series. John Benjamins, pp. 21–30.

[3] Roland Barthes. *Mythologies*. Translated by Annette Lavers. London: Paladin, 1972.

[4] Kemal Mert Başoğlu. "The 'Barren' I: On the Loss of Subjectivity and Fragmentation of Consciousness in Samuel R. Delany's Babel-17". Graduation project, Istanbul Kültür University. 2019.

[5] Marie-France Beauvois. "Optic aphasia: A process of interaction between vision and language". In: *Philosophical Transactions of the Royal Society of London, Series B* 298 (1982), pp. 35–47.

[6] Geert Brône, Kurt Feyaerts, and Tony Veale, eds. *Cognitive Linguistics and Humor Research*. Vol. 26. Applications of Cognitive Linguistics. De Gruyter, 2015.

[7] Michel-Antoine Burnier and Patrick Rambaud. *Le Roland-Barthes sans peine*. Balland, 1978.

[8] Aifric Campbell. *The Semantics of Murder*. London: Serpent's Taill, 2008.

[9] Ronnie Cann. *Formal semantics. An introduction*. Cambridge Textbooks in Linguistics. Cambridge University Press, 1993.

[10] Daniel Chandler. *Semiotics, the basics*. Routledge, 2002.

[11] Paul Cobley and Litza Jansz. *Introducing Semiotics. A Graphic Guide*. Icon Books Ltd, 2012.

[12] Allan M. Collins and M. Ross Quillian. "Retrieval time from semantic memory". In: *Journal of Verbal Learning and Verbal Behavior* 8 (1969), pp. 240–247.

[13] Sarah Coulson. *Semantic Leaps. Frame-Shifting and Conceptual Blending in Meaning Construction*. Cambridge University Press, 2001.

[14] Wilhelm Dindorf. *Menandri Fragmenta*. Paris: Ambroise-Firmin Didot, 1846.

[15] Chet Van Duzer. *Sea Monsters on Medieval and Renaissance Maps*. British Library, 2013.

[16] Johannes Fahndrich. "Semantic Decomposition and Marker Passing in an Artificial Representation of Meaning". PhD thesis. Technische Universität Berlin, 2018.

[17] Fernanda Ferreira and Charles Clifton. "The independence of syntactic processing". In: *Journal of Memory and Language* 25 (1986), pp. 348–368.

[9] https://alis.lojban.org

[18] Richard Fikes and Tom Kehler. "The Role of Frame-Based Representation in Reasoning". In: *Communications of the ACM* 28 (1985), pp. 904–920.

[19] Charles J. Fillmore. "Scenes-and-frames semantics". In: *Linguistic Structures Processing*. Ed. by Antonio Zampolli. Amsterdam: North-Holland, pp. 55–81.

[20] Charles J. Fillmore. "The case for case". In: *Universals in linguistic theory*. Ed. by Emmon Bach and Robert T. Harms. New York: Holt, Rinehart, and Winston, 1968, pp. 1–88.

[21] Charles J. Fillmore and Beryl T. Atkins. "Towards a Frame-Based Lexicon: The Semantics of RISK and its Neighbors". In: *Frames, Fields, and Contrasts*. Ed. by Adrienne Lehrer and E. Feder Kittay. Lawrence Erlbaum, 1992, pp. 75–102.

[22] Jerry A. Fodor. *The Modularity of Mind*. The MIT Press, 1983.

[23] Bernhard Ganter, Gerd Stumme, and Rudolf Wille, eds. *Formal Concept Analysis: Foundations and Applications*. Vol. 3626. Springer LNAI. 2005.

[24] Jérôme Garcin, ed. *Nouvelles Mythologies*. Éditions du Seuil, 2014.

[25] Dirk Geeraerts. *Theories of Lexical Semantics*. Oxford University Press, 2010.

[26] Dirk Geeraerts and Hubert Cuyckens, eds. *The Oxford Handbook of Cognitive Linguistics*. Oxford University Press, 2007.

[27] Rachel Giora. *On Our Mind: Salience, Context, and Figurative Language*. Oxford University Press, 2003.

[28] Jacob W. Glazier and Timothy J. Beck. "Apocalypse, language, temporality: an alien encounter in Ted Chiang's "Story of Your Life"". In: *Subjectivity* 11 (2018), pp. 267–284.

[29] Cliff Goddard and Anna Wierzbicka. "Introducing Lexical Primitives". In: *Semantic and Lexical Universals*. Ed. by Cliff Goddard and Anna Wierzbicka. Amsterdam/Philadelphia: John Benjamins Publishing Company, 1994.

[30] Cliff Goddard and Anna Wierzbicka. *Words and Meanings*. Oxford University Press, 2014.

[31] Paul Grice. *Studies in the Way of Words*. Harvard University Press, 1989.

[32] Pierre Guiraud. "Les Champs morpho-sémantiques (critères externes et critères internes en étymologie)". In: *Bulletin de la société de linguistique de Paris* 52 (1956), pp. 265–288.

[33] Petr Hájek, Jeff Paris, and John Shepherdson. "The Liar Paradox and Fuzzy Logic". In: *The Journal of Symbolic Logic* 65 (), pp. 339–346.

[34] Trevor A. Harley. "Speech errors: Psycholinguistic approach". In: *The Encyclopaedia of Language and Linguistics*. Ed. by K. Brown. Vol. 11. Oxford: Elsevier, 2006, pp. 739–744.

[35] Trevor A. Harley. *The Psychology of Language: From Data to Theory*. 4th ed. Psychology Press, 2014.

[36] Jisup Hong, Elise Stickles, and Ellen Dodge. "The MetaNet metaphor repository: Formalized representation and analysis of conceptual metaphor networks". In: *Proceedings of the 12th International Cognitive Linguistics Conference (ICLC 12)*. 2013.

[37] Yan Huang. *Pragmatics*. Oxford Textbooks in Linguistics. Oxford University Press, 2015.

[38] Yan Huang, ed. *The Oxford Handbook in Pragmatics*. Oxford University Press, 2017.

[39] Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, 2002.

[40] Michael Jacobson. "On Asemic Writing". In: *Asymptote* (July 2013).

[41] Fairouz Kamareddine and Twan Laan. *A Reflection on Russell's Ramified Types and Kripke's Hierarchy of Truths*. Computing Science Report 95/18. Eindhoven, 1995.

[42] Hans Kamp, Josef van Genabith, and Uwe Reyle. *Discourse Representation Theory*. Vol. 15. Handbook of Philosophical Logic. Springer, 2011, pp. 125–394.

[43] Hans Kamp, Josef van Genabith, and Uwe Reyle. "Discourse Representation Theory". In: *Semantic Theories*. Ed. by Claudia Maienborn, Klaus von Heusigner, and Paul Portner. de Gruyter, 2019, pp. 321–384.

[44] Christine Kettaneh. "Mute Melodies". In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applica-

tions. Brest: Fluxus Editions, 2020, pp. 561–577. DOI: https://doi.org/10.36824/2020-graf-kett.

[45] Fahad Khan et al. “Generative Lexicon Theory and Linguistic Linked Open Data”. In: *Proceedings of the 6th International Conference on Generative Approaches to the Lexicon (GL2013)*. 2013.

[46] David Kronenfeld and Gabriella Rundblad. “The semantic structure of lexical fields: Variation and change”. In: *Words in Time*. Mouton de Gruyter, 2003, pp. 67–114.

[47] George Lakoff. “Hedges: A Study in Meaning Criteria”. In: *Journal of Philosophical Logic* 2 (1973), pp. 458–508.

[48] George Lakoff. *Women, Fire, and Dangerous Things. What Categories Reveal about the Mind*. The University of Chicago Press, 1987.

[49] George Lakoff and Mark Johnson. *Metaphors we live by*. The University of Chicago Press, 1980.

[50] Cognitive Linguistics – Foundations of Language. Ed. by Ewa Dąbrowska and Dagmar Divjak. De Gruyter Mouton, 2019.

[51] Adrienne Lehrer. *Semantic fields and lexical structure*. Amsterdam: North-Holland, 1974.

[52] Cognitive Linguistics – A Survey of Linguistic Subfields. Ed. by Ewa Dąbrowska and Dagmar Divjak. De Gruyter Mouton, 2019.

[53] Carl Linnæus. *Systema naturæ per regna tria naturæ, secundum classes, ordines, genera, species, cum characteribus, differentiis, synonymis, locis*. Stockholm: Laurentius Salvius, 1758.

[54] John Lyons. *Introduction to theoretical linguistics*. Cambridge: Cambridge University Press, 1968.

[55] John Lyons. *Linguistics Semantics. An Introduction*. Cambridge: Cambridge University Press, 1995.

[56] Hagit Magen and Asher Cohen. “Modularity beyond perception: evidence from single task interference paradigms”. In: *Cogn Psychol.* 55 (2007), pp. 1–36.

[57] William C. Mann and Sandra A. Thompson. “Rhetorical Structure Theory: Toward a function theory of text organization”. In: *Text* 8 (1988), pp. 243–281.

[58] Daniel Marcu. *The Theory and Practice of Discourse Parsing and Summarization*. The MIT Press, 2000.

[59] Michael McCloskey and Sam Glucksberg. “Natural categories: Well-defined or fuzzy sets?” In: *Memory & Cognition* 6 (1978), pp. 462–472.

[60] Marvin Minsky. “A framework for representing knowledge”. In: *The Psychology of Computer Vision*. Ed. by P. Winston. New York: McGraw-Hill, 1975, pp. 211–280.

[61] Marvin Minsky. “Jokes and their Relation to the Cognitive Unconscious”. In: *Cognitive Constraints on Communication*. Ed. by Lucia Vaina and Jaakko Hintikka. Vol. 18. Studies in Linguistics and Philosophy. Springer, 1984, pp. 175–200.

[62] Bernhard Nebel and Gert Smolka. “Attributive description formalisms ... and the rest of the world”. In: *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence Final Report on the IBM Germany LILOG-Project*. Ed. by Otthein Herzog and Claus-Rainer Rollinger. Springer, 1991, pp. 439–452.

[63] Mark Newman. *Networks. An Introduction*. 2nd ed. Oxford University Press, 2018.

[64] Winfried Nöth. *Handbook of Semiotics*. Advances in Semiotics. Indiana University Press, 1990.

[65] Øystein Ore. “Galois Connexions”. In: *Transactions of the American Mathematical Society* 55.3 (1944), pp. 493–513.

[66] Claudio Paolucci. “The distinction between semantics and pragmatics: The point of view of semiotics”. In: *Intercultural Pragmatics* 18.3 (2021), pp. 293–307.

[67] Walter Porzig. “Wesenhafte Bedeutungsbeziehungen”. In: *Beiträge zur Geschichte der deutschen Sprache und Literatur* 58 (1934), pp. 70–97.

[68] Uta E. Priss. “Classification of meronymy by methods of relational concept analysis”. In: *Proceedings of the 1996 Midwest Artificial Intelligence Conf., Bloomington, Indiana*. 1996.

[69] Uta E. Priss. “The Formalization of WordNet by Methods of Relational Concept Analysis”. In: *WordNet. An Electronic Lexical Database*. Ed. by Christiane Fellbaum. MIT Press, 1998, pp. 179–196.

[70] James Pustejovsky. *The Generative Lexicon*. The MIT Press, 1996.

[71] M. Ross Quillian. *Semantic memory*. Cambridge, MA: Bolt Beranak and Newman, 1966.

[72] Christian Rathke. “Object-oriented Programming and Frame-based Knowledge Representation”. In: *Proceedings of the IEEE International Conference on Tools with AI*. 1993, pp. 95–98.

[73] Nick Riemer. *Introducing Semantics*. Cambridge University Press, 2010.

[74] L. David Ritchie. “Frame-Shifting in Humor and Irony”. Communication Faculty Publications and Presentations. Paper 14. http://archives.pdx.edu/ds/psu/8752. 2005.

[75] Eleanor Rosch. “Cognitive representations of semantic categories”. In: *Journal of Experimental Psychology* 104 (1975), pp. 192–223.

[76] Eleanor Rosch. “On the internal structure of perceptual and semantic categories”. In: *Cognitive development and the acquisition of language*. Ed. by T. E. Moore. New York: Academic Press, 1973, pp. 111–144.

[77] Eleanor Rosch. “Principles of categorization”. In: *Cognition and Categorization*. Ed. by Eleanor Rosch and Barbara B. Lloyd. Hillsdale, NJ: Larence Erlbaum Associates, 1978, pp. 27–48.

[78] Laurent Roussarie. *Sémantique formelle. Volume 1: Introduction à la grammaire de Montague*. Vol. 4. Textbooks in Language Sciences. Language Science Press, 2017.

[79] Charlotte Rozé. “Vers une algèbre des relations de discours”. PhD thesis. Université Paris Diderot-Sorbonne, 2013.

[80] Josef Ruppenhofer et al. “FrameNet II: Extended Theory and Practice”. https://framenet2.icsi.berkeley.edu/docs/r1.7/book.pdf. 2016.

[81] “Das sprachliche Feld: eine Auseinandersetzung”. In: *Wortfeldforschung. Zur Geschichte und Theorie des sprachlichen Feldes*. Ed. by Ludwig Erich Schmidt. Vol. 10. Neue Jahrbücher für Wissenschaft und Jugendbildung. Darmstadt: Wissenschaftliche Buchgesellschaft, pp. 428–429.

[82] Peter Schwenger. *Asemic. The Art of Writing*. University of Minnesota Press, 2019.

[83] Kate Scott. *Pragmatics in English*. Cambridge University Press, 2023.

[84] Thomas Sebeok. *Signs: An Introduction to Semiotics*. 2nd ed. University of Toronto Press, 2001.

[85] George Edgar Slusser. *Robert A. Heinlein, Stranger in His Own Land*. San Bernardino, CA: Newcastle/Borgo Press Original, 1976.

[86] Manfred Stede, Maite Taboada, and Debopam Das. “Annotation Guidelines for Rhetorical Structure”. http://www.sfu.ca/~mtaboada/docs/research/RST_Annotation_Guidelines.pdf. 2017.

[87] Mark Steedman. *The Syntactic Process*. The MIT Press, 2000.

[88] John Ridley Stroop. “Studies of interference in serial verbal reactions”. In: *Journal of Experimental Psychology* 18 (1935), pp. 643–662.

[89] Darko Suvin. *Metamoprhoses of Science Fiction*. Yale University Press, 1979.

[90] Alfred Tarksi. “The Semantic Conception of Truth: and the Foundations of Semantics”. In: *Philosophy and Phenomenological Research* 4 (1944), pp. 341–376.

[91] Philip Thody and Piero. *Introducing Barthes. A Graphic Guide*. Icon Books Ltd, 2013.

[92] Cognitive Linguistics – Key Topics. Ed. by Ewa Dąbrowska and Dagmar Divjak. De Gruyter Mouton, 2019.

[93] Peter Pericles Trifonas. *International Handbook of Semiotics*. Springer, 2015.

[94] Tony Veale, Ekaterina Shutova, and Beata Beigman Klebanov. *Metaphor. A Computational Perspective*. Morgan & Claypool, 2016.

[95] Piek Vossen, Francis Bond, and John McCrae. “Toward a truly multilingual GlobalWordnet Grid”. In: *Proceedings of the 8th Global WordNet Conference (GWC)*. Global Wordnet Association, 2016, pp. 424–431.

[96] Paul Warren. *Introducing Psycholinguistics*. Cambridge University Press, 2013.

[97] Bonnie L. Webber. "Discourse deixis: Reference to discourse segments". In: *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistic (ACL-88)*. State University of New York at Buffalo, NY, 1988, pp. 113–122.

[98] Anna Wierzbicka. *Imprisoned in English. The Hazards of English as a Default Language*. Oxford University Press, 2014.

[99] Anna Wierzbicka. *Semantic Primes and Universals*. Oxford University Press, 1996.

[100] Lotfi A. Zadeh. "Fuzzy sets". In: *Information and Control* 8 (1965), pp. 338–353.

[101] Lotfi A. Zadeh. "Liar's Paradox and Truth-Qualification Principle". University of California Memorandum No. UCB/ERL M79/34 https://www2.eecs.berkeley.edu/Pubs/TechRpts/1979/ERL-m-79--34.pdf. 1979.

[102] Edward N. Zalta. "Gottlob Frege". Stanford Encyclopedia of Philosophy, https://plato.stanford.edu/entries/frege. 2022.

Chapter 7
Controlled Natural Languages

In the preceding chapters, we have underlined the complexity of language and the problems one faces when analyzing it using mathematical tools. As we mentioned when discussing grammaticality and acceptability, sometimes people prefer acceptable rather than grammatical linguistic productions so that any analyzer based on strict grammatical rules will fail. ☞ 3

We can legitimately ask: What is the point of having logical approaches to language when language is inherently a mess?

The good news is that for some tasks, one can define one's language as a subset of a natural language and use as much complexity as one needs, but not more. Such languages are called *controlled natural languages* (CNL) and have existed for almost a century. Kuhn [7] defines a CNL as follows:

1. It is based on exactly one natural language (its "base language").
2. The most crucial difference between it and its base language (but not necessarily the only one) is that it is more restrictive concerning lexicon, syntax, and/or semantics.
3. It preserves most of the natural properties of its base language so that speakers of the base language can intuitively and correctly understand texts in the controlled natural language, at least to a substantial degree.
4. It is a constructed language, which means that it is explicitly and consciously defined, and is *not* the product of an implicit and natural process, even though it is based on a natural language that *is* the product of an implicit and natural process.

Conditions 2 and 3 exclude programming languages, and Condition 1 excludes artificial languages such as Esperanto or Lojban. According to Schwitter [11], the goals of CNLs can be of three types: ☞ 75

1. to improve communication among humans, especially speakers with different native languages;
2. to improve manual, computer-aided, semi-automatic, or automatic translation; and
3. to provide a natural and intuitive representation for formal notations.

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_7

It is hard to compare CNLs. Kuhn [7] has introduced a classification scheme called PENS, based on four parameters:

1. *Precision* $P^{1\leq i\leq 5}$, with possible values: imprecise, less imprecise, reliably interpretable, deterministically interpretable, with fixed semantics. This is about the dependency of the CNL on context, background knowledge, heuristics, and the like. At levels 4 and 5, we have CNLs defined by formal grammars. ☞ 247
2. *Expressiveness* $E^{1\leq i\leq 5}$, with possible values: inexpressive, low expressiveness, high expressiveness, maximal expressiveness. Kuhn [7] further adds some criteria that allow evaluation of the E parameter of CNLs:

   (a) universal quantification over individuals,
   (b) relations of arity greater than 1,
   (c) general rule structures (if–then statements with multiple universal quantification that can target all argument positions of relations)
   (d) negation,
   (e) general second-order universal quantification over concepts and relations,
   (f) support for existential quantification,
   (g) equality, and
   (h) types of supported speech acts (such as declarative, interrogative, directive, and indirect speech acts).

   ☞ 277 If propositional logic, description logics, FOL, and second-order predicate calculus were CNLs (which they are not since they do not have a base language), they would be of expressiveness E^1, E^2, E^3, and E^4, respectively.
3. *Naturalness* $N^{1\leq i\leq 5}$, with possible values: unnatural, with dominant unnatural elements, with dominant natural elements, with natural sentences, with natural texts. This is about form (use of symbols, brackets, keywords) and the proximity with the base language.
4. *Simplicity* $S^{1\leq i\leq 5}$, with possible values: very complex, without exhaustive descriptions, lengthy descriptions, short descriptions, concise descriptions. This is about the effort needed to fully implement the syntax and semantics in a mathematical model, such as a computer program. S^1 and S^2 are typically defined by *proscriptive rules* (what is forbidden, for the base language) and S^3, S^4, S^5 by *prescriptive rules* (what is allowed).

In this scheme, a natural language would be $P^1E^5N^5S^1$. The Shadok language (the language of the creatures of an animated television series created by French cartoonist Jacques Rouxel and broadcasted in the late sixties and early seventies [5], a syllabic language with only four syllables: Ga, Bu, Zo, Meu) would be $P^1E^1N^2S^5$ (but again, one can debate whether Shadok has French as its base language or not).

Let us consider some examples of CNLs.

7.1 Simplifications of English: Basic English, Simple English, and Caterpillar English

Basic English [10] is one of the oldest CNLs: it was introduced in 1930 by Ogden, a philosopher, psychologist, linguist, editor, art critic, antiquarian bookseller, antique dealer, and expert on musical boxes, author among other things of the influential *The Meaning of Meaning* (1923). Fifty years before the BASIC (= "Beginners' All-purpose Symbolic Instruction Code") programming language, Ogden's term "Basic" was already an acronym for "Business Academic Scientific International Commercial." Basic English is a drastic simplification of English since its vocabulary contains only 850 nouns and 18 verbs, the complete list of which is "put," "take," "give," "get," "come," "go," "make," "keep," "let," "do," "be," "seem," "have," "may," "will," "say," "see," and "send". As an example, here are the first four sentences of the Gettysburg Address in its original form, and in Basic English:

ORIGINAL	BASIC ENGLISH
Fourscore and seven years ago our fathers brought forth, on this continent, a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived, and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting-place for those who here gave their lives, that that nation might live.	Seven and eighty years have gone by from the day when our fathers gave to this land a new nation—a nation which came to birth in the thought that all men are free, a nation given up to the idea that all men are equal. Now we are fighting in a great war among ourselves, putting it to the test if that nation, or any nation of such a birth and with such a history, is able long to keep united. We are together on the field of a great event in that war. We have come to give a part of that field as a last resting-place for those who went to their death so that that nation might go on living.

Notice the need to replace the nation "so dedicated [that] can long endure" with "with such a history, is able long to keep united," which has hardly the same meaning. But this is unavoidable if we consider the restrictions on nouns and verbs.

The *Simple English* Wikipedia project [3] claims to use Basic English. As of May 2023, it contains 230,829 articles, about thirty times less than standard English Wikipedia, ten times less than French Wikipedia, and roughly as much as Greek Wikipedia. Well understood, writing texts on 230,829 different subjects with only 850 nouns and 18 verbs is impossible. Let us compare the first sentences of standard English and Simple English Wikipedia pages on "Philosophy":

STANDARD ENGLISH WIKIPEDIA	SIMPLE ENGLISH WIKIPEDIA
Philosophy (from Greek: φιλοσοφία, philosophia, 'love of wisdom') is the systematized study of general and fundamental questions, such as those about existence, reason, knowledge, values, mind, and language.	Philosophy is the study of underlying things. It tries to understand the reasons or basis for things. It also tries to understand how things should be. "Philosophia" is the Ancient Greek word for the "love of wisdom".

Standard Wikipedia gives the Greek term (with transcription and translation), uses the fancy term "systematized study," and enumerates thematic areas using abstract terms such as "reason," "knowledge," and "mind." The Simple English version thrice uses the word "things" to avoid abstraction, with the risk of becoming unintelligible. And what are "underlying things" in the first place?

Caterpillar Fundamental English (CFE) [1] was a CNL used between 1971 and 1982 by Caterpillar Inc., a heavy equipment manufacturing company headquartered in Illinois. CFE had a vocabulary of about 850 words, avoiding polysemy and ambiguity. The idea was to teach personnel of Caterpillar CFE through a 30-lesson course so that they could translate the 20,000 Caterpillar publications into 50 different languages. In 1982, CFE was abandoned because teaching it to uneducated personnel had a high failure rate. New markets in countries using other writing systems increased the challenge even more. CFE was replaced by *Caterpillar Technical English* [4], a CNL developed in collaboration with Carnegie Mellon University. Vocabulary has been expanded from about 850 to 70,000 (!) words and lexical and syntax parsers have been developed to allow on-the-fly validation during the authoring process. The goal was, of course, to obtain automatic translations into the 50 target languages, but it is unclear whether this goal has been achieved.

In the PENS classification scheme, all CNLs of this section can be classified as $P^2E^5N^5S^1$, i.e., they differ from English only in the P value: less impressive for the former and imprecise for the latter.

7.2 Formalizable Controlled Languages: Attempto Controlled English, PENG

PENG (*Processable English*) [11][1] is a CNL that can be automatically translated via discourse representation structures into FOL with equality. PENG has been specifically designed to write specifications and use cases. Its latest version, PENG$^{\text{ASP}}$ [13], is bidirectional: its grammar can be used to translate a specification into an executable ASP program and to generate a semantically equivalent verbalization of that ASP program, where ASP (Answer Set Programming) is a knowledge representation language for non-monotonic reasoning based on Prolog [9]. Here is an example text in PENG$^{\text{ASP}}$ and its translation in ASP:

☞ 307

> COMP3160 and COMP3220 are units. Liam is a student and Olivia is a student. Every student is either enrolled in COMP3160 or is enrolled in COMP3220. If a student withdraws from a unit then the student is not enrolled in that unit. It is not the case that Liam is enrolled in COMP3220. Olivia withdraws from COMP3160. Who is enrolled in COMP3220?

```
named(1, comp3160). class(1, unit).
named(2, comp3220). class(2, unit).
named(3, liam). class(3, student).
named(4, olivia). class(4, student).
```

[1] The title of this article, *English as a formal specification language* is, of course, a pun on Montague's 1970 seminal paper *English as a formal language*.

```
1 { prop(X, 1, enrolled_in) ; prop(X, 2, enrolled_in) } 1
:- class(X, student).
-prop(X, Y, enrolled_in) :- class(X, student),
pred(X, Y, withdraw_from), class(Y, unit).
:- prop(3, 2, enrolled_in).
pred(4, 1, withdraw_from).
answer(PN) :- named(X, PN), prop(X, 2, enrolled_in).
```

In the PENS classification scheme, PENG can be classified as $P^5E^3N^4S^3$.

Attempto Controlled English (ACE) is a CNL with an automatic and unambiguous translation into first-order logic. ACE was originally intended to specify software, but has afterward been used as a general knowledge representation language [6] in several application domains and, in particular, for the Semantic Web.

Here is an example of a small paragraph written in Attempto Controlled English:

> A customer enters a card and a code. If the code is valid then SimpleMat accepts the card. If the code is not valid then SimpleMat rejects the card.

together with the first analysis step (a paraphrase with variables assigned to entities):

> There is a customer X1. The customer X1 enters a code X2 and a card X3. If the code X2 is valid then SimpleMat accepts the card X3. If it is false that the code X2 is valid then SimpleMat rejects the card X3.

For more Attempto Controlled English examples, see also Exercises 6-1, 9-3, 10-1, and 10-2, on p. 203, 307, 338, and 338.

In the PENS classification scheme, Attempto Controlled English can be classified as $P^4E^3N^4S^3$, i.e., slightly less precise than PENG$^{\mathrm{ASP}}$. The former is deterministically interpretable, while the latter has fixed semantics.

It is no coincidence that PENG and ACE are quite similar. Rolf Schwitter, the developer of PENG, was Norbert Fuchs' PhD student in Zürich and his 1998 Ph.D. thesis [12] was based on ACE. He then emigrated to Macquarie University in Australia where he started developing PENG in 2002, following the Attempto tradition.

7.3 A CNL for Mathematics: ForTheL

We all know that mathematics requires absolute rigor, but we forget that mathematics is communicated through mathematical text, i.e., a mixture of natural language and formulas. The exact ratio of formulas vs. text depends on the author's style. Compare the following sentences:

> Let there be an integer i that is strictly positive and less than five.
> Let there be i strictly positive and less than five.
> Let there be i such that $0 < i \leq 5$.
> Let there be $0 < i \leq 5$.

The first sentence is the most complete. The second one is underspecified: it relies on the fact that the notation "i" is used mainly for integers and not for rationals or reals. The third uses a formula instead of words. The fourth is ambiguous: what is the object of the verb “be”? Is it the entire inequality, only i, or number 0? Despite its ambiguity, one can often see this formulation in mathematical texts because 0 and 5 are not variables, and the expression “let there be” is expected to be followed by one or more variables. Here, i is the only variable, so necessarily, “let there be” can only be applied to i.

The text/formulae ratio of a text and the slight ambiguities as the one we just mentioned are of no (or little) importance to mathematicians. Still, they can be problematic for parsing mathematics by machines. Some standardization of mathematical text can, therefore, be helpful.

But there is another way of doing mathematics: one can let a machine prove theorems, as this has already been the case for the well-known Four-Color Theorem: four colors suffice to color a planar map so that no edge connects areas of the same color. This theorem was submitted to De Morgan in 1852. It was finally proved more than a century later, in 1975, on the IBM 360 computer of the University of Illinois (a machine with an astonishing RAM of 9MB!) [2].

But for a machine to parse mathematics, it has to be communicated to the machine in a parseable form. Since the seventies, several research teams started rewriting mathematics in various formal languages from its very foundations. These languages are entirely parseable by machines but are not very attractive to humans, requiring a stiff learning curve.

ForTheL (Formula Theory Language) [8] as part of the *Naproche* system (Natural language proof checking) is a CNL and provides a readable but standard syntax for the verbal part of the mathematical text.

☞ 178 To analyze mathematical text, Naproche uses not DRSs (see p. 178) but PRSs (proof representation structures). PRSs are like DRSs except that the referents’ part of the box is split into two: the left part contains discourse referents (which are integers), and the right part contains mathematical referents (mathematical expressions, whether variables or formulas). Here is a small text in Naproche:

```
Define $m$ to be square iff there is an integer $n$ such that $m=n^2$
```

from which, by compiling the LaTeX part, we get the obvious human-readable result

Define m to be square iff there is an integer n such that $m = n^2$

The same input is parsed in PRSs as follows:

$$\begin{array}{|l|l|}\hline 1 & \mathrm{m} \\ \hline \multicolumn{2}{|l|}{\text{math\_id}(1,m)} \\ \multicolumn{2}{|l|}{\text{square}(1)} \\ \hline \end{array} := \begin{array}{|l|l|}\hline 2,3 & n,\ m = n^2 \\ \hline \multicolumn{2}{|l|}{\text{math\_id}(2,n)} \\ \multicolumn{2}{|l|}{\text{math\_id}(3,m = n^2)} \\ \multicolumn{2}{|l|}{\text{holds}(3)} \\ \hline \end{array}$$

Mathematical formulas remain unchanged and are assigned to discourse referents. Furthermore, “square” and “holds” are predicates, the arguments of which are discourse referents. After conversion into PRSs, the Naproche system will type-check the result and forward the whole to a reasoner, thereby validating or invalidating the proof provided by the mathematician. So Naproche is not only a CNL for writing mathematics in a standardized way but also a proof checker.

7.4 Exercises

Exercise 6-1: Discovery of Attempto Controlled English

ChatGPT provided us with a "playful classification of feline mischiefs," as follows:

> The *high-level perpetrator* masters the art of knocking objects off shelves and tables with precision timing.
> The *sudden leaping expert* demonstrates an extraordinary ability to launch themselves onto unsuspecting humans from hidden corners.
> The *yarn connoisseur* possesses a keen taste for unraveling balls of yarn into intricate and bewildering patterns.

Translate them into ACE[2] (Attempto Controlled English). ☞ 201

Exercise 6-2: How simple is Simple English Wikipedia?

Choose randomly a thousand Simple English Wikipedia pages with corresponding Standard English Wikipedia pages and compare them.

Exercise 6-3: Write haikus inspired by themes by Emily Dickinson in Python

Haiku can be considered a kind of controlled language (in the sense of constrained writing as cultivated by the Oulipo). Your task: Write a Python program that will generate random haikus based on themes of poems by Emily Dickinson. ☞ 241

Exercise 6-4: Do Daleks use a controlled language?

> Vision impaired! Sabotage! The Doctor is still alive! Exterminate!
>
> Daleks in *The Power of the Doctor* S39E9

Daleks, the Doctor's biggest enemy, and a prominent Nazi metaphor were introduced to the British audience on December 28th, 1963. They already mentioned extermination at that time, but the first iconic "Exterminate!" outcry was uttered on May 22, 1965. Their use of language has the reputation of being terse, choppy, and even mechanical. But is it controlled? In other words, does it obey specific lexical and syntactic constraints? How does it compare to the language of other Whovian characters, e.g., companions?

[2] The name "ACE" is inspired by the 7th Doctor's companion Ace (1987–1989).[3]

[3] The note 2 just above is a typical example of fake news, as the author of Attempto has confirmed to us (private communication).

References

[1] Eric Adolphson, Teruko Mitamura, and Eric Nyberg. "Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English". In: *Proceedings of the Second International Workshop on Controlled Language Applications (CLAW'98), Pittsburgh, PA*. 1998, pp. 1–12.

[2] Kenneth Appel and Wolfgang Haken. "The Solution of the Four-Color-Map Problem". In: *Scientific American* 237 (1977), pp. 108–121.

[3] William Coster and David Kauchak. "Simple English Wikipedia: A New Text Simplification Task". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 665–669.

[4] Stephen Crabbe. "Controlled languages for technical writing and translation". In: *Proceedings of the 9th Portsmouth Translation Conference*. Portsmouth, 2009, pp. 48–62.

[5] Sébastien Denis. *Les Shadoks: Histoire, esthétique et 'pataphysique*. INA, 2016.

[6] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. "Attempto Controlled English for knowledge representation". In: *Proceedings of Reasoning Web—4th International Summer School 2008*. Ed. by Cristina Baroglio et al. Vol. 5224. Lecture Notes in Computer Science. 2008, pp. 104–124.

[7] Tobias Kuhn. "A Survey and Classification of Controlled Natural Languages". In: *Computational Linguistics* 40 (2014), pp. 121–170.

[8] Adrian De Lon et al. "The Isabelle/Naproche Natural Language Proof Assistant". In: *Proceedings of CADE 2021*. Ed. by A. Platzer and G. Sutcliffe. Vol. 12699. Lecture Notes in Artificial Intelligence. 2021, pp. 614–624.

[9] Gelfond Michael and Kahl Yulia. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[10] Charles K. Ogden. *Basic English: A general introduction with rules and grammar*. London: Paul Treber & Co., 1930.

[11] Rolf Schwitter. "English as a formal specification language". In: *Proceedings of the 13th International Workshop on Database and Expert Systems Applications*. 2002, pp. 228–232.

[12] Rolf Schwitter. "Kontrolliertes Englisch für Anforderungsspezifikationen". PhD thesis. Universität Zürich, 1998.

[13] Rolf Schwitter. "The Grammar of PENG ASP Explained". In: *Proceedings of the Seventh International Workshop on Controlled Natural Language (CNL 2020/21)*. 2021.

Part II
Mathematical Tools

The so-called *symbolic approach* to Natural Language Processing consists in describing the various linguistic layers (morphology, syntax, meaning) by using mathematical tools. Graphs, i.e., sets of binary relations, can be used, e.g., to represent words and their relations. Formal languages, i.e., subsets of a free monoid, are used to model morphology and syntax. Logic, a millenia-old discipline, on which the foundations of mathematics are based, is used to represent the meaning of sentences and of discourse, for a fragment of natural language or for controlled natural languages. Combining graphs and logic, we get powerful tools for representing information and knowledge: ontologies and conceptual graphs. The main difference between the content of this part and that of Part IV, “Statistical Methods,” is explainability: all methods of this part are transparent, all results can be explained. In the trade-off between transparency and performance, we favor the former.

Chapter 8
Graphs

8.1 Definitions

Graph theory, as a branch of mathematics, is characterized by the simplicity of its elementary object, the *graph*, and its extraordinary amount of applications. A graph consists of *vertices* and *edges*. Vertices are arbitrary objects, and edges are binary relations between vertices. This definition may sound like a simplification, but it is not. It *is* the real thing.

More formally, if V is an arbitrary set (called the set of *vertices*) and if E (the set of *edges*) is a subset of $V \times V$, then the pair $G = (V, E)$ is a graph. For an edge $(v_1, v_2) \in E$, where $v_1, v_2 \in V$, we will call v_1 its predecessor and v_2 its successor.

In some textbooks, there is a distinct terminology for *directed graphs* (also called *digraphs*, they are the general case) and *non-directed graphs* (the case where there is no distinction between edge (v_1, v_2) and edge (v_2, v_1), for $v_1, v_2 \in V$). In the former case, some authors call the edges "arcs." To simplify the terminology, we will call "edges" both the directed and the undirected pairs of vertices.

Graphs can be studied at three different levels: the global level (the whole graph), the local level (a single vertex and its surroundings), as well as a third, intermediate level, the level of *paths*. A *path* is a sequence of adjacent edges (*adjacent*, meaning that if e_1 and e_2 are consecutive edges, then the successor of e_1 is the predecessor of e_2). Paths are intermediate in that they can be pretty local (for example, a single vertex connecting with its neighbor) or, in some cases, pretty global, like a single path that goes through all vertices of a graph.

Let us define some basic notions, starting with the local ones:

- a vertex v_1 is a *neighbor* of a vertex v_2 when $v_1 \neq v_2$ and there is an edge (v_1, v_2) or (v_2, v_1) between them;
- the *degree* of a vertex is the number of its neighbors. We can separate the *incoming degree* (the incoming degree of v_1 is the number of edges (v_2, v_1)) from the *outcoming degree* (the outcoming degree of v_1 is the number of edges (v_1, v_2));
- when the predecessor is the same as the successor, then we have a special case of edge called a *loop edge*;

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_8

- often, we will attach additional information to vertices or edges. If this information is numeric, we will call it a *weight*, but it can be of any kind. For example, it can be a label, as the words and intermediate symbols of formal grammars are labels of nodes of syntax trees.

On the intermediate level, we define:

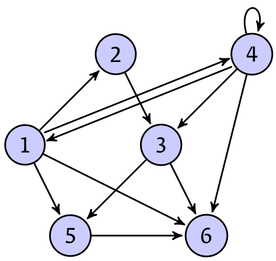
- a *cycle* as being a path $e_1, \ldots, e_n$ in which the predecessor of e_1 is also the successor of e_n;
- a graph with no cycles is called *acyclic*;
- a *trail* (or *simple path*) as a path, the edges of which are pairwise distinct (i.e., a path that does not contain any cycles).

On the global level, we define:

- *connecticity* as a global property of a graph: a graph is *connected* if for every pair of vertices v_1, v_2 there is a path between v_1 and v_2;
- *connect components* as maximal subgraphs that are connected;
- a graph with no edges is called *discrete*. In a discrete graph, all vertices are connected components. When a graph is connected, it has only a single connected component.

In essence, a graph is just a set of vertices and a set of binary relations between vertices, called edges. Nevertheless, a mathematical object proves useful in representing graphs, namely *matrices*.

Given a digraph G containing n vertices $v_1, \ldots, v_n$, its *adjacency matrix* A is a $n \times n$ matrix such that its cell $a_{i,j}$ at row i and column j contains value 1 when there is an edge from v_i to v_j, otherwise the value of the cell is 0. On the diagonal of the matrix, we find loop edges. In Fig. 8.1, the reader can see an example of a graph and the corresponding adjacency matrix.



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 8.1 A digraph and its adjacency matrix

One of the most popular Python packages for graph processing is *networkx*. Here is the code for storing the graph of Fig. 8.1 in memory:

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([(1,2),(1,4),(1,5),(1,6),(2,3),(3,5),(3,6),(4,1),\
    (4,3),(4,4),(4,6),(5,6)])
```

When G is non-directed, we act as if we had both an edge from v_i to v_j and an edge from v_j to v_i. Our matrix is then symmetric.

An exciting feature of adjacency matrices is that their eigenvalues are all zero when the graph contains no cycle. This gives us a simple way of verifying whether a graph is acyclic. This result comes from the fact that if we take the n-th product of an adjacency matrix by itself, A^n, the value of each cell i, j represents the number of paths of length n between v_i and v_j. It follows that on the diagonal, we get the number of cycles of length n. If the graph is acyclic, then the diagonals of all matrices A^n will be zero, and the eigenvalues of a matrix with zero diagonal are also zero. It has been shown that the two conditions are equivalent: a graph is acyclic iff the eigenvalues of its adjacency matrix are all zero.

Besides acyclicity, adjacency matrices have another interesting feature related to connected components. If we call D the matrix that contains in each cell $d_{i,i}$ of its diagonal the degree of vertex v_i and zero elsewhere, then the matrix $A - D$ (called the *Laplacian* of the graph) has as many zero eigenvalues as the graph has connected components.

A *bipartite* graph is a graph G, the vertices of which belong to two partitions P_1 and P_2, in a way that every edge connects the two partitions. In other words, no edge is internal to a partition. A typical example would be a graph with students and skills, with edges representing the fact that a student has acquired a skill. In this case, drawing edges between students (a student will not acquire another student) or between skills (a skill does not have skills) makes no sense. See §11.2 for another example of bipartite graphs: conceptual graphs.

8.1.1 Trees

A very important special kind of graph is *trees*. A *tree* is an acyclic and connected graph. Being a tree is an equilibrium situation between two extreme states: the one of the discrete graph (no edges at all) and the one of the complete graph (as many edges as possible). Except for very small graphs (with less than three vertices), completeness and acyclicity are mutually exclusive. We can move from one extreme to another by removing edges from a complete graph or by adding edges to a discrete graph. At some moment, we may reach a state where both conditions (acyclicity and connectedness) co-exist: at that moment, we have a tree. Trees are very fragile: adding or removing a single edge will break their treeness.

Trees have very interesting properties:

- a tree with n vertices has exactly $n - 1$ edges,
- there is a unique path connecting two arbitrary vertices of a tree,
- a tree with n vertices has between 2 and $n - 1$ vertices of degree 1, called *leaves* (the former case is the one of a *linear* tree: a path where each vertex is of degree 2 expect for the two extremities that are of degree 1; the latter case is the one of a *star-graph* where the center vertex has degree $n - 1$ and all others degree 1).

Contrarily to trees in the real world, mathematical trees do not need a root. Nevertheless, one uses *rooted* trees in many applications. Once a tree has a root, it automatically becomes *directed*: as there are unique paths from the root to all vertices of the tree, one takes the direction induced by these paths. The reader should be aware of the fact that although directed graphs, in general, are drawn using arrows to indicate directions, the edges of rooted trees are not represented by arrows since the direction is evident once the root is known. Often, rooted trees are represented in such a way that the root is on the top of the diagram, and the direction of edges corresponds to the physical downward direction of the diagram. As an example, take any syntax tree in this book.

In a rooted tree, we have an additional interesting property:

- every vertex (except for the root) has a single predecessor, called the *parent*.

A *forest* is a tree without the connectivity condition, in other words, an acyclic graph that is not necessarily connected.

8.2 Basic Graph Algorithms

In this section, we will define the basic algorithms for searching a graph, searching the shortest paths, calculating the centrality of each node, and identifying communities. We will not provide the pseudo-code of each algorithm—the interested reader may find these in book references in the "Further Reading" section.

8.2.1 Search in a Graph

There are two basic "search" algorithms: *breadth-first search* (BFS) and *depth-first search* (DFS). Interestingly, they are called "search" algorithms in English and German ("Suche"$_{\text{DE}}$) and "route" ("parcours"$_{\text{FR}}$) algorithms in French: the methods are the same, but in the former case, one stops once a node with given properties is found, while in the latter the whole graph is covered.

Both BFS and DFS allow the user to walk through all vertices[1] using edges. The difficulty lies in the fact that in the case of a cycle, one should consider every edge at most one time and avoid endless loops.

Intuitively, DFS and BFS behave in symmetric ways:

- DFS will move away from the starting vertex, randomly choosing vertices among the current node's neighbors. This will go on until the most recent node has no neighbors, in which case the algorithm goes backward and considers all the unvisited neighbors;

[1] It is essential to realize that one walks through all vertices but *not* through all edges.

- BFS will consider all the neighbors of the starting vertex, then the (unvisited) neighbors of the neighbors, and so on. In other words, BFS advances by concentric circles around the starting vertex.

DFS may be considered an "aggressive" approach ("go as far away as possible and then start covering the surroundings") and BFS as a "conservative approach" ("enlarge your neighborhood a step at a time"). In Fig. 8.2, the reader can see the same graph search by DFS (on the left) and by BFS (on the right), the labels on vertices representing the order of selection of vertices. Notice that vertex 32 (the last one visited) is very close to vertex 01 in DFS and very far from it in BFS.

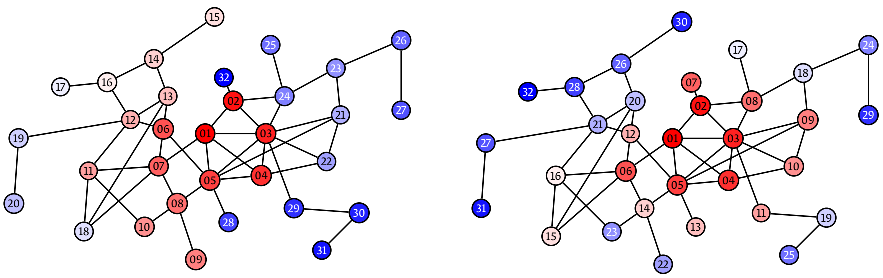


Fig. 8.2 The same graph search through the DFS algorithm (on the left) and the BFS algorithm (on the right). Labels represent the selection order of vertices. Colors go from red for 01, to blue for 32.

From the programmer's point of view, DFS is implemented using a recursive procedure, and BFS uses a FIFO (first-in, first-out) stack.

While BFS and DFS are equal in complexity, BFS has a significant advantage: vertices are visited in order of distance from the starting vertex, so if the user of the algorithm is looking for a vertex that has a given property and is *closest to the starting vertex*, then BFS is the best choice.

The *networkx* methods for BFS and DFS are `edge_bfs(G)` and `edge_dfs(G)`, respectively, where `G` is a graph object.

8.2.2 Shortest Paths

In our description of BFS, we mentioned a vertex "closest to the starting vertex." By "closest," we meant considering the distance obtained by counting edges. It is an integer distance, where every edge counts as 1. In most applications, what is used is not the number of edges but a weight attached to edges (for example, the geographic distance measured in km or miles on a map). BFS will provide the shortest path when counting edges, but another algorithm is needed when distance is based on weights attached to edges, a "shortest path search algorithm." The most popular algorithm in this family is *Dijkstra's algorithm*. It is fast, easy to implement, and has only one constraint: the weights attached to edges must be positive.

Dijsktra's algorithm will calculate all shortest paths between a given starting vertex and every other vertex of the graph. By doing so, the shortest path between the starting vertex and a specific other vertex may be calculated early or at the very last moment. The algorithm does not allow prioritizing some part of the graph so that a path with a given end vertex is calculated as quickly as possible. If speed is an issue (it depends on the size of the data and the available computing power), then an enhanced version of Dijkstra's algorithm, called *A* algorithm*, allows prioritization of some part of the graph for shortest path search. For this, one has to define a *heuristic function*, a numeric function of edges that decreases when one is in the right region of the graph or moving in the right direction. Intuitively if we want to calculate the shortest path between Kansas City and Los Angeles, there is no need to calculate the shortest path between Kansas City and New York first so that the heuristic function will be minimal for paths going westwards from Kansas City and maximal for paths going in north- or south- or eastwards.

One way to create a weighted graph in *networkx* is to use the method

```
G.add_weighted_edges_from(L)
```

where `L` is a list of tuples, each tuple containing the edge's predecessor, successor, and weight. The *networkx* method for the Dijkstra algorithm is

```
nx.all_pairs_dijkstra(G)
```

It returns two dictionaries for each vertex: the first contains the distance to each other reachable vertex, and the other is the path to reach each other. There is also a method for the A* algorithm: `astar_path(G,a,b)`, where `G` is a graph, `a` is the source vertex and `b` is the target vertex. It returns the shortest path between `a` and `b`. When a heuristic `h` is available, the method can be called as `astar_path(G,a,b,heuristic=h)`. In this case, `h` must be a function with two node labels as input and a numeric value as output.

8.2.3 An Example: Word Ladders

> October 24, 1879.
> My dear Kathleen,—I was really pleased to get your letter [...] I hope I may now count you as one of my child-friends. I am fond of children (except boys)
> If you can do "Doublets," with how many links do you turn KATH into LEEN?
> Your affectionate friend,
> Charles L. Dodgson
>
> Lewis Carroll, *Letter to Kathleen Eschwege*

Here is an example that will allow us to use the notions and algorithms we have seen so far. It is not a typical NLP project but just a simple application of graphs to a real-life problem.

Word ladders are a game used to acquire a language's vocabulary: provided two words A and B of equal length, the challenge is to find intermediate words, differing by a single letter at a time, that progressively transform A into B, one letter at a time. Lewis Carroll claims to have invented this game in 1878. Donald Knuth devotes three chapters of his *Selected Papers on Fun and Games* [14, pp. 415–427] to it.

Let us see how to use graph theory to obtain word ladders. To make the problem a bit more NLP-ish, let us add some constraints:

(a) we would like minimal ladders,
(b) we would like ladders that require the least cognitive load.

First, we need a corpus of words, preferably in order of decreasing frequency (to respect constraint (b)). For that, we can take frequency lists obtained from Project Gutenberg and provided on the Wiktionary site[2]. Let us take the 40,000-word lists as of April 16, 2006. It looks like this:

```
the   56271872
of    33950064
and   29944184
...
zipper 7.91134
zloty  7.91134
zoetrope   7.91134
```

where "złoty" is the Polish currency and "zoetrope" is a cylindrical device based on the principle of cinematography, where one looks through slots and see images moving when the cylinder spins.

Cognitive load is a function of the "difficulty" of words. The difficulty is correlated to frequency (a word like "zoetrope" may be beautiful but requires significantly more effort to be accessed in the mental lexicon, than a word like "together" which ☞ 78 is the first 8-letter word of the list, at position 278). But frequency alone may not be a good criterion for difficulty. Let us call the list we obtained L_1.

[2] https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists

A commonly used criterion for the difficulty is the grade level at which the word is taught in primary school. We have used a list of 3,240 words ordered by their introduction in the curriculum, from the first week of first grade to the 36th week of grade 6.[3] It looks like this:

```
the    1
with   1
what   1
...
practically  6
pamphlet     6
thoroughly   6
```

We merged the two lists by giving priority to L_2: we obtain a new list L by the formula $L = L_2 \cup L_1 \setminus (L_1 \cap L_2)$, where the order of a word in L is its order in L_2 if it belongs to this list, or its order in $L_1 \setminus (L_1 \cap L_2)$ plus the cardinal of L_2 (i.e., if the last word in L_2 is the N-th word of L, then the first word of $L_1 \setminus (L_1 \cap L_2)$ is the $(N+1)$-th word of L).

We create a directed graph, the vertices of which are words from our list, all of the same length. We have taken words of length 4 and thereby obtained 2,786 vertices.

For each word w we take all possible character strings obtained by changing a single letter and look them up in the graph. When the string is indeed an existing word w', we create two edges: one from w to w' and one from w' to w. We assign weights to these two edges as follows: the weight of the former is the order of w' in L, and the weight of the latter is the order of w in L. I.e., each edge has a weight equal to the order of its successor in L.

After this step, our graph of four-letter words has 23,176 edges. This means that the average outgoing degree of a word is 8.32. There are 108 words of degree 0, i.e., words that cannot be transformed into any other English four-letter word, the simplest ones being "upon," "once," "evil" (3rd grade), "view" (4th grade), "rely" (5th grade) and "obey," "ugly," "owns," "Ohio," "envy" (not in L_2). Knuth [10, p. 22] calls these words, *aloof words* ("aloof" meaning "reserved, physically or emotionally distant," and aloof being itself an aloof word, and therefore an *autological word*). ☞ 74

The highest (outgoing) degree is 29 and the only word of degree 29 is "mare": it can be transformed into "make," "more," "made" (1st grade), "care" (2nd grade), "bare," "hare" (3rd grade) and "mark," "mary," "mere," "dare," "rare," "male," "mate," "fare," "pare," "tare," "ware," "mire," "mace," "mage," "mane," "mase," "maze," "Marc," "mari," "marl," "Mars," "mart," "maru" (not in L_2).

To build a word ladder between words w and w' we apply Dijkstra's shortest path search algorithm with w as the starting vertex and stop when we have found a shortest path between w and w'. If we use a constant weight of 1 for each edge, we get minimal-length ladders; when using weights from L (e.g., the inverse of the rank of a word in the list L) we get an approximation of minimal cognitive effort ladders. If w' is never attained by the algorithm, it means that w and w' are in different connected components, and no ladder can exist between them.

[3] https://www.k12reader.com/first-grade-spelling-words/

Here are some examples of word ladders calculated by our algorithm:

- between "what" and "they": When we ignore word frequencies, we get the minimal-length ladder

  what–whet–whey–they.

  This ladder contains two difficult words: "(to) whet," i.e., to stimulate, and "whey," the liquid remaining after milk has been curdled and strained in the process of making cheese. Searching for the ladder of minimum cognitive load, we get

  what–that–than–then–when,

  a ladder of length 5 with only first-grade words;
- an example where the ladder length ratio is extreme: for words "were" and "will," we get the minimal-length ladder of length 4

  were–wire–wile–will,

  containing one difficult word: "(to) wile," meaning "to lure." The ladder of minimal cognitive load is much longer:

  were–here–herd–head–heat–beat–belt–bell–well–will,

  it consists of 10 words taught in grades 1 and 2.

8.2.4 Processing WordNet as a Graph

WordNet has a graph structure, which we illustrate[4] on Fig. 8.3, as rendered by *Gephi* ☞ 143
software[5] [1]. How can we process it as a graph, using, for example, *networkx*?

That is quite easy:

```
import networkx as nx
from nltk.corpus import wordnet as wn

G=nx.DiGraph()

for synset in list(wn.all_synsets('n')):
    if not G.has_node(synset.name()):
        G.add_node(synset.name())

for synset in list(wn.all_synsets('n')):
    for hyp in synset.hypernyms():
        if not G.has_edge(synset.name(),hyp.name()):
            G.add_edge(synset.name(),hyp.name())
```

[4] Figure 8.3 of this book is a low resolution bitmap. The reader interested in the "real thing," that is, the PDF file containing *all* 74,374 synsets and 75,834 relations, can download it from the following address: `http://www.fluxus-virus.com/fichiers/wordnet_graph.pdf`.

[5] `https://gephi.org/`

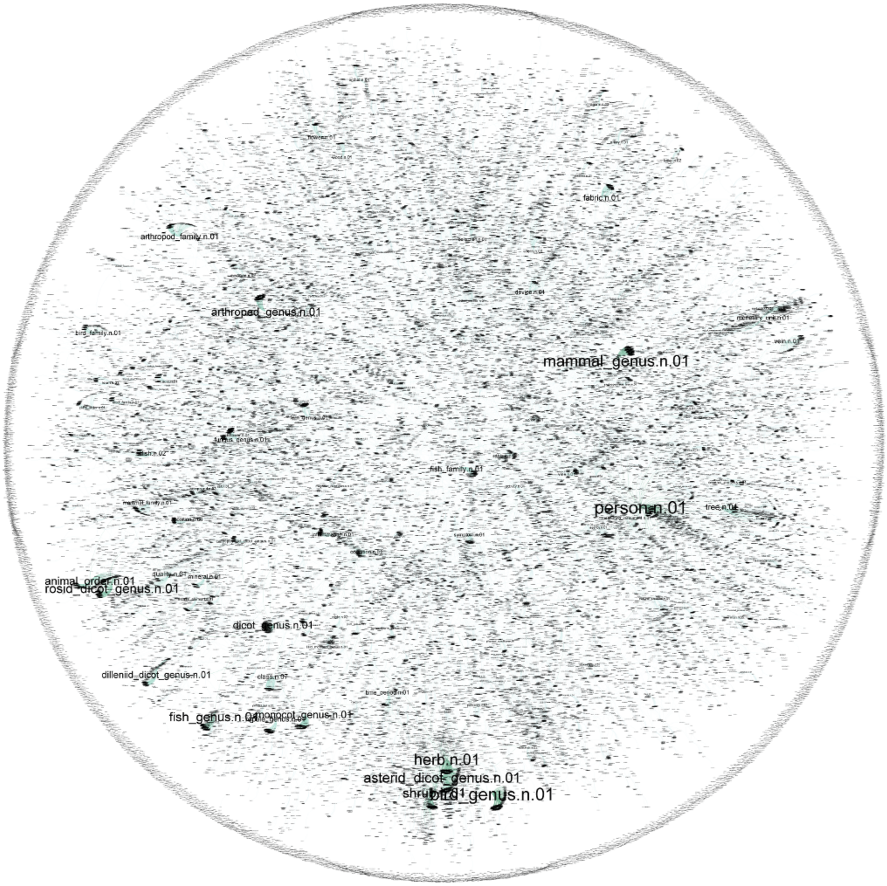


Fig. 8.3 WordNet's graph rendered by *Gephi* software, using Yifan Hu's Scalable Force Directed Placement rendering algorithm [12] with label size representing degree. The synsets visible to the naked eye are those of very high degree (for example, "person$_1$" (aka `person.n.01`) has a degree of 404).

```
14
15  print(G.size(),"edges",G.number_of_nodes(),"nodes")
```

In the first loop, lines 6–8, we create as many nodes as noun synsets in WordNet. Then, on lines 11–13, we add edges for all hyperonym relations, provided they are not already created. The operation is quite fast. The label of the node is the "name" of the synset, such as `car.n.01` for "car$_1$".

Let us now do something impossible without using a graph theory algorithm. Let us calculate the *diameter* of WordNet, together with the corresponding path.

The diameter of a graph is the *length of the longest shortest path*. This may seem contradictory, but it is not. Between each pair of nodes v_1, v_2 located in the same connected component, we have a shortest path. This path may not be unique, but its length is unique. It is the number of edges between v_1 and v_2. The diameter is the distance between the nodes farthest from each other. It is the length of the longest shortest path among the shortest paths of all vertex pairs. Finding the diameter of WordNet will provide us with a pair of synsets that are semantically the most unrelated. Will it be "quasar$_1$" and "crochet$_1$"? (as in p. 413, where we will ask ChatGPT to build a sentence with these two words), or maybe "football$_2$" and "intelligence$_1$"? (just joking!).

We need to calculate the shortest paths of all pairs to find the diameter. If we restrict ourselves to nouns only (the largest part of WordNet, which is not connected to the other parts: verbs, adjectives, and adverbs), we get a graph of 82,115 nodes. That makes 6,742,791,110 pairs of distinct nodes. If, for example, one second for each pair, we will need 213.8 years to complete the task. That is far less than the 7.5 million years needed by Deep Thought in *h2g2* to calculate the answer to the Ultimate Question of Life, the Universe, and Everything (which is 42). However, it is still a long time, and it is way beyond the computer's LEB (Life Expectancy at Birth), and ours as well. We need another plan.

Fortunately, *networkx* has a method for that:

```
path = dict(nx.all_pairs_shortest_path(G))
```

This method calculates the shortest paths of all pairs simultaneously. Such an approach can save time because we have a monotonicity property. If $v_1, v_2, \ldots, v_n$ is a shortest path between v_1 and v_n, then every subpath of it is also a shortest path (otherwise it would mean we have found a short-cut and $v_1, v_2, \ldots, v_n$ wasn't the shortest path in the first place).

We have launched the script with this command. The following day the computer was still running without a hint of a result. So we sought a faster solution. Looking on *StackOverflow* (an excellent way to find answers to any problem), we found a piece of advice by Suuungi[6] and followed it. This advice was to use another graph theory package called *graph-tool* to search for all shortest paths. Indeed, *graph-tool* is very fast. It needs to be installed first on the system level and then as a Python package via *conda*[7]. Here is the code that resulted from the common use of *networkx* and *graph-tool*:

```
import graph_tool.all as gt
import graph_tool.topology as gtop
from nltk.corpus import wordnet as wn
import networkx as nx

G_gt = gt.Graph(directed=False)
G=nx.Graph()

for synset in list(wn.all_synsets('n')):
    if not G.has_node(synset.name()):
        G.add_node(synset.name())
```

6 https://stackoverflow.com/questions/69649566/how-do-i-speed-up-all-pairs-dijkstra-path-length

7 See https://git.skewed.de/count0/graph-tool/-/wikis/installation-instructions for installation instructions.

```
12
13 for synset in list(wn.all_synsets('n')):
14     for hyp in synset.hypernyms():
15         if not G.has_node(hyp.name()):
16             G.add_node(hyp.name())
17         if not G.has_edge(synset.name(),hyp.name()):
18             G.add_edge(synset.name(),hyp.name())
19
20 vertices = {}
21 inverse_vertices={}
22 for node in G.nodes:
23     v = G_gt.add_vertex()
24     vertices[node] = v
25     inverse_vertices[v]=node
26
27 for src, dst in G.edges():
28     e = G_gt.add_edge(vertices[src], vertices[dst])
29
30 res=gtop.shortest_distance(G_gt)
31
32 MAX=0
33 max_v=0
34 max_i=0
35 for v in G_gt.vertices():
36     for i in range(len(res[v])):
37         if res[v][i] < 2147483647:
38             if res[v][i] > MAX:
39                 MAX=res[v][i]
40                 max_v=v
41                 max_i=i
42 print(MAX,inverse_vertices[max_v],inverse_vertices[max_i])
```

On line 7 we instantiate a *graph-tool* graph `G_gt`, on lines 21–26 we copy `G`'s vertices into `G_gt`. As *graph-tool* uses only numeric labels, we need a dictionary to return to synset names once the calculations are completed. This is the purpose of line 25. On lines 27–28 we copy edges from `G` to `G_gt`. Line 30 is a *graph-tool* method that will calculate short paths for all vertex pairs. The result is a dictionary with vertices as keys and dictionaries as values. These dictionaries have vertices again as keys and distances as values. On lines 35–41, we go through the entire structure and keep the first shortest path of maximum length. Line 37 is to avoid "infinite" values that occur when the two vertices are not in the same connected component.

Here is the result obtained after 1h59′18″ of calculations on a MacBook Pro 2,3 GHz Intel Core i9 8 cores with 32 MB of RAM: the diameter of WordNet is 30. The first pair of synsets at a distance equal to a diameter has been

exposure_therapy$_1$ leather_carp$_1$

If the reader does not know these terms (it was the case of the author), there is no need to google them. It suffices to take the path between them to understand what they stand for:

exposure_therapy$_1$ desensitization_technique$_1$ behavior_therapy$_1$ psychotherapy$_2$ therapy$_1$ medical_care$_1$ treatment$_1$ care$_1$ work$_1$ activity$_1$ act$_2$ event$_1$ psychological_feature$_1$ abstraction$_6$

entity$_1$ physical_entity$_1$ causal_agent$_1$ person$_1$ organism$_1$ animal$_1$ young$_1$ young_fish$_1$ fish$_1$ bony_fish$_1$ teleost_fish$_1$ soft-finned_fish$_1$ cypriniform_fish$_1$ cyprinid$_1$ carp$_2$ domestic_carp$_1$ leather_carp$_1$

We realize that "exposure therapy" is a kind of psychotherapy and that "leather carp" is a domestic carp. We did google the latter to find out that it is one of those carps we see in ponds, which can grow up to 50 kg. Looking at the path, we see two knowledge domains that have contributed to WordNet: psychology and biology. The former has a maximal depth of 14 and the latter of 16, so the path length is 30. Another necessary condition for these domains to provide a diameter path is that their concepts are completely disjoint: from the psychology side, we climb up to "entity$_1$" by a ladder of abstractions, and on the biology side, we climb down to "leather_carp$_1$" by a ladder of living entities.

As the careful reader has undoubtedly noticed, we have cheated. On lines 6 and 7 of the code, we took instances of undirected graphs, even though hyperonyms are directed relations. We did this to facilitate the calculation of the shortest paths. The result is that when going down from "entity$_1$" to "leather_carp$_1$," all relations are not hyperonymic. Some of them are hyponymic: we first go down to "person," but then go up again to "organism" and down again to "animal." Otherwise, one would conclude that a carp is a kind of person. Considering a carp as a person may be a gratifying experience, but, as a hyperonymic relation, it is not contained in WordNet.

8.3 Vertex Centrality

We may want to compare the "importance" of vertices with respect to the graph structure. We have double-quoted the word "importance" because there are many ways of defining this concept. We will consider some of them, having the common feature that they can be numerically calculated from the graph's data only (and not by importing data from other sources).

In the following, we will call importance *centrality*. We will introduce three types of centrality: degree centrality (and its improvements, eigenvector centrality, and PageRank), closeness centrality, and betweenness centrality. We will illustrate each one by finding the most central synsets in WordNet.

8.3.1 Degree Centrality

In the word-ladder example, the vertex of *highest degree* was the one of the word "mare." Does this make "mare" an essential word in the English language? Not necessarily, but in some cases, a vertex is considered "more central" when it has a higher degree than others. After all, don't we measure the notoriety of a movie star or a politician by their number of Facebook followers? Considering Facebook accounts as vertices and friendships as edges, the number of friends is exactly that: the degree of the account vertex.

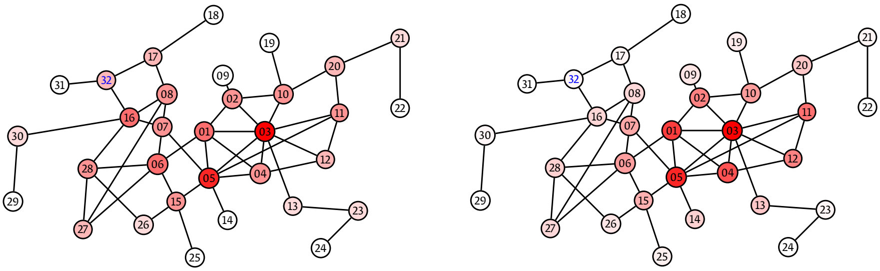


Fig. 8.4 On the left: degree centrality of a graph, represented in vertex shades from white (centrality 0) to red (centrality 1); on the right: eigenvector centrality calculated on the same graph.

We can start by proclaiming the vertex of the highest degree to be of centrality 1, the one of the least degree of centrality 0, and all others to have centralities between 0 and 1, with respect to their degrees. In the left part of Fig. 8.4, the reader can see the degree centrality of a graph represented by shades between white and red. As expected, vertices 03 and 05 are the most central.

They are surrounded by less central vertices, such as 02, 10, 20, 11, 12, 04, etc., and at the periphery of the graph, we have vertices of centrality 0, such as 18, 09, 19, 22, 24, etc.

There is a problem, though: in the figure, vertex 14 is a neighbor of vertex 05, the most central vertex of the graph. If edges represent "friendship," then 05 is 14's only (and hence, best) friend. One would expect 14 to inherit some of 05's notoriety—as so nicely illustrated in the movie *Notting Hill* (1999), where the unknown bookseller, played by Hugh Grant, becomes the lover of a famous Hollywood actress played by Julia Roberts. The paparazzi discover the former at some point, and he inherits, willy-nilly, some of the notoriety of the latter.

How can we improve our centrality measure to take inheritance into account? A method called *eigenvector centrality* uses the adjacency matrix's eigenvalues. In the right part of Fig. 8.4, the reader can see eigenvector centrality calculated on the same example as on the left side. As can be seen, by the shades of red, 14 indeed inherits centrality from 05. On the contrary, nodes 32 and 17 are now less central since they may be of degree 3, but their neighbors are not very central. In some sense, degree centrality corresponds to a society where every person with n friends is equally important, no matter where she is in the graph. In contrast, eigenvector centrality corresponds to the case where people in the periphery will never become central because they live among non-central people…

These centrality methods are available in the *networkx* package:

```
degree_centrality(G)
eigenvector_centrality(G)
```

where `G` is a graph. They return a dictionary with vertices as keys and centrality values as values.

But what happens with directed graphs? (Friendship on Facebook may be undirected, but other social networks such as Twitter or Instagram need to be modeled using digraphs since on one side of an edge, one has a follower and on the other a tweeter.)

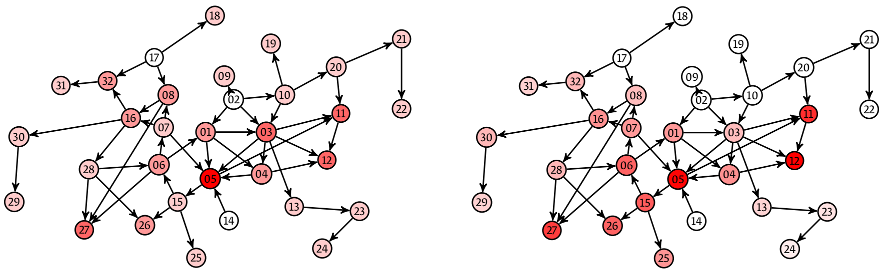


Fig. 8.5 On the left: degree centrality of a digraph, represented in vertex shades from white (centrality 0) to red (centrality 1); on the right: eigenvector centrality calculated on the same digraph.

The calculation of degree centrality and of eigenvector centrality can be done separately for ingoing and outgoing edges. We then obtain *indegree centrality* and *outdegree centrality*. In Fig. 8.5, the reader can see the indegree centrality (degree centrality on the left and eigenvector centrality on the right) of a graph represented by shades between white and red.

As the reader will have immediately noticed in the example, indegree eigenvector centrality is a disaster: not only does inheritance fail in the case of vertices 14 and 05, but an entire part of the graph (vertices 09, 02, 10, 19, 20, 21, 22) is of centrality 0. Why does this happen? Indegree eigenvector centrality can be non-zero only for vertices with ingoing edges. Therefore, 02 has no centrality at all. Inheritance works only in the direction of edges, so 02 can get no centrality from 01 or 03. And since 02 has no centrality, it cannot give any centrality to 09 and 10, which, in turn, cannot give any centrality to 20, and so on.

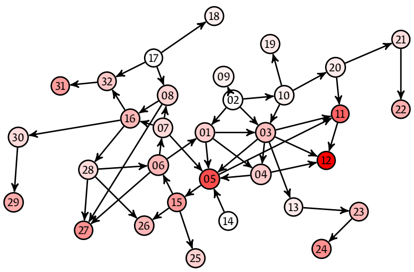


Fig. 8.6 PageRank centrality of the same graph as in Fig 8.5.

To solve this problem, Katz introduced a variant of eigenvector centrality in 1953 [13] that carries his name. An additional refinement of Katz's centrality became the famous *PageRank* algorithm [18], the fundamental ingredient of the Google search engine. Katz's idea was to "give to the poor" by attaching a small amount of centrality to each vertex with an incoming edge, whether the predecessor of the edge has any centrality or not. This attachment is considered when eigenvectors are calculated and gives much smoother results. As shown in Fig. 8.6, only vertices with no incoming edge, such as 14, have zero centrality.

The PageRank centrality can also be calculated using *random walks*. Let us consider an agent moving from one vertex to another in the following way: if the current node has neighbors, there is a probability α that the agent moves to one of the neighbors of the current node (with equal probability among all neighbors) and a probability $1 - \alpha$ that the agent does a *teleport*, i.e., a jump to some other vertex (with equal probability among all vertices of the graph). Here, α is a value in $[0, 1]$ (for $\alpha = 0$ the agent will only jump, so the structure of the graph is not taken into account; for $\alpha = 1$ the agent will remain in the connected component of the starting vertex). As announced in [5] (the paper that announced the Google search engine) and beautifully explained in [11], the relative amount of time the agent passes on each vertex is equivalent to the PageRank centrality of the vertex.

The digraph centrality methods mentioned above are all available in the *networkx* package:

```
in_degree_centrality(G)
out_degree_centrality(G)
eigenvector_centrality(G)
katz_centrality(G)
pagerank(G,alpha=0.85)
```

where `G` is a digraph, and `alpha` is PageRank's α parameter. They return a dictionary with vertices as keys and centrality values as values.

Degree Centrality in WordNet

To measure centrality in WordNet, we first reduce the size of the graph by taking only the largest connected component:

```
import networkx as nx
from nltk.corpus import wordnet as wn
import numpy
import io,sys

G=nx.Graph()

for synset in list(wn.all_synsets('n')):
    if not G.has_node(synset.name()):
        G.add_node(synset.name())

for synset in list(wn.all_synsets('n')):
    for hyp in synset.hypernyms():
```

```
14            if not G.has_node(hyp.name()):
15                G.add_node(hyp.name())
16            if not G.has_edge(synset.name(),hyp.name()):
17                G.add_edge(synset.name(),hyp.name())
18
19    largest_cc = max(nx.connected_components(G), key=len)
20    S = G.subgraph(largest_cc).copy()
21
22    degr=nx.degree_centrality(S)
23
24    degr_keys=sorted(list(degr.keys()),key=lambda x:degr[x],reverse=True)
25    print([(x,degr[x]) for x in degr_keys[:100]])
```

WordNet is not connected. It has 7,726 connected components (out of which 7,714 are singletons!). On line 19, we take the largest connected component (the comparison key being the number of nodes); on line 20, we take the subgraph of `G` induced by this component. The largest component has 74,374 nodes (90.5% of all nodes) and 75,834 edges (only 16 edges of the global graph are not in the largest component). On line 22, we calculate degree centrality. The method returns a dictionary with the nodes as keys and their centralities as values. On line 25, we sort the keys according to decreasing centrality. Here are the first ten results:

person$_1$, bird_genus$_1$, mammal_genus$_1$, herb$_1$, asterid_dicot_genus$_1$, shrub$_1$, rosid_dicot_genus$_1$, fish_genus$_1$, animal_order$_1$, arthropod_genus$_1$

This is no surprise since these are precisely the synsets visible in Fig. 8.3. The first one is the hyperonym of all persons mentioned in WordNet (404 of them), and all the others are genera in biology.

We have also calculated PageRank centrality and got the following:

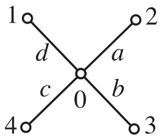
bird_genus$_1$, mammal_genus$_1$, person$_1$, asterid_dicot_genus$_1$, herb$_1$, rosid_dicot_genus$_1$, fish_genus$_1$, shrub$_1$, arthropod_genus$_1$, animal_order$_1$

It is six of one and half a dozen of the other. We get the same synsets in a slightly different order. For example, "person$_1$" now ranks 3rd, while it ranked 1st in degree centrality. Why did this happen? Birds have many subdivisions, so a bird genus is close to some other subdivision, which also has a high degree and, therefore, is central. Having central neighbors increases PageRank centrality but leaves degree centrality unaffected. This is probably the case of persons who are not part of a biological taxonomy and, therefore, do not have neighbors as central as birds and mammals.

8.3.2 Closeness Centrality

Closeness centrality is a "geometric" kind of centrality. When we consider each vertex of a graph as a starting vertex for Dijkstra's algorithm and obtain the shortest paths to all other vertices, we also obtain the distances between this starting vertex and all other vertices (distances calculated either by counting the number of edges of each shortest path or by taking the sum of weights of the edges of each shortest path). Suppose we have the distances $d(v, w)$ between a given starting vertex v and all other vertices w. In that case, we can also calculate the *average distance* $\ell(v) = \sum_{w\in V} \frac{d(v,w)}{\#\{w\in V, w\neq v\}}$ between a given starting vertex v and all other vertices w of the graph, where $\#\{w \in V, w \neq v\}$ is the number of vertices other than v. This calculation requires connectivity, so it can only be performed inside a connected component.

The idea behind closeness centrality is that the more a vertex v is central, the more $\ell(v)$ will be small. Here is a trivial example that illustrates this fact:



It is a graph of vertices $\{0, 1, 2, 3, 4\}$ and edges $\{a, b, c, d\}$. The sum of distances between 0 and all other vertices is $a+b+c+d$, so the average distance is $\frac{a+b+c+d}{4}$. The sum of distances between 1 and all other vertices is $d+a+d+b+d+c+d = a+b+c+4d$, and therefore the average distance is $\frac{a+b+c}{4} + d$, since we always have to use edge d first to access other vertices. In this example, 0 is the most central vertex since it has the smallest average distance.

Obviously, centrality has to increase when a vertex is more central. Therefore, average distance is rather the opposite of a centrality measure. A first definition of *closeness centrality* is to take $C(v) = \frac{1}{\ell(v)} = \frac{1}{\sum_w d(v,w)}$. Still, one quickly finds out that this definition has some important drawbacks, the simplest one being that when the graph is not connected, some of the $d(v, w)$ in the denominator will be infinite. Hence, $C(v)$ will be systematically zero. To solve that problem, we take $C'(v) = \frac{1}{n-1} \sum_{w\neq v} \frac{1}{d(v,w)}$, using the *harmonic average distance*. We call this centrality, *harmonic closeness centrality*.

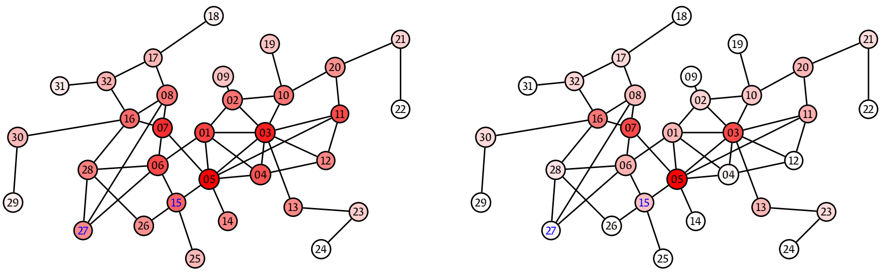


Fig. 8.7 Closeness centrality (on the left) and betweenness centrality (on the right) of the same graph as in Fig 8.5.

In the left part of Fig. 8.7, the reader can see the centrality measure applied to the graph of Fig. 8.5. The result is quite intuitive: vertices 01, 03, 04, 05, and 07 are the most central, surrounded by less central vertices, and finally, nodes at the outer extremities of paths like 22 and 24 are the least central.

These methods are available in the *networkx* package:

```
closeness_centrality(G)
harmonic_centrality(G)
```

where `G` is a graph. They return a dictionary with vertices as keys and centrality values as values.

Closeness Centrality in WordNet

To measure closeness centrality, we only change the measure's name in the code. The calculation is significantly longer than for degree centrality and PageRank, which was almost instant. Our a MacBook Pro 2,3 GHz Intel Core i9 8 cores with 32 MB of RAM took 2h7′4″ to finish the task. Here are the results:

abstraction_6, physical_entity_1, entity_1, group_1, causal_agent_1, object_1, $\text{psychological\_feature}_1$, relation_1, attribute_2, communication_2

There is no surprise. What we find are the top elements of WordNet's hierarchy. This shows that WordNet is well-balanced since the highest nodes in the hierarchy are equidistant from all nodes.

8.3.3 Betweenness Centrality

The idea of betweenness centrality is that a vertex is central when you often cross it in the graph. The extreme case is one of two subgraphs G_1, G_2 connected to a given vertex v as follows: G_1—v—G_2. To travel from one graph to the other, one must necessarily go through vertex v, which has a high betweenness centrality.

In the right part of Fig. 8.7, the reader can see the betweenness measure applied to the graph of Fig. 8.5. The result is somewhat surprising: although vertices 05, 07, and 03 are still very central, vertices 06, 01, and 02 that lie between them are significantly less central, and vertex 04 is almost of centrality 0. On the other hand, as expected, all vertices of degree 1 are of zero centrality since they are never used as in-between vertices for traveling between parts of the graph.

This method is available in the *networkx* package:

```
betweenness_centrality(G)
```

where `G` is a graph. It returns a dictionary with vertices as keys and centrality values as values.

Betweenness Centrality in WordNet

To measure betweenness centrality in WordNet, we started by simply changing the measure's name in the code. About six hours later, we stopped the process and did a trick. The trick was to remove all degree-1 nodes. This makes a huge difference since there are 56,751 nodes of degree 1, which is 76.3% of the largest component. What allows us to do so? It is the fact that a degree-1 node v has only a single neighbor v', and the betweenness centrality of v will be the same as the one of v' since all paths connecting v to the rest of the graph will necessarily cross v'. The node centralities of the reduced graph will numerically not be the same as the ones of the original graph, but we are only interested in the top-10 ranking. One can argue that if there is a degree-1 node v next to the most central node v_{max} in the original graph, then v is probably also among the top-ranked, and reducing the graph has eliminated v. We reply that a degree-1 node that happens to be next to v_{max} may be top-ranked as well but is *not interesting* since it gives us no insight into the structure of the graph.

Therefore, the top 10 of the reduced graph will give us essential information on the betweenness centrality of the non-reduced graph. After reducing the graph (17,646 vertices and 19,106 edges), the calculation took only 15′14″. Here are the results:

abstraction$_6$, physical_entity$_1$, person$_1$, psychological_feature$_1$, artifact$_1$, entity$_1$, causal_agent$_1$, attribute$_2$, organism$_1$, event$_1$

The list is almost similar to the list of top-ranked nodes for closeness centrality. Why does this happen? The reduced graph has 17,646 vertices and 19,106 edges and is connected (since we have only removed degree-1 nodes). This means that there are only 1,460 edges that introduce cycles (because a tree of 17,646 vertices would have exactly 17,645 edges). 1,460 edges are only 8% of the total amount of edges, i.e., the reduced graph is more-or-less tree-like. In a tree, betweenness centrality and closeness centrality are correlated since all paths are unique, and therefore, when a node v is closeness-central and, therefore, in the center of a bunch of nodes, it means that many paths connecting these nodes have to cross v since there is no other way.

8.4 Community Detection

We all know the phenomenon of communities emerging in social networks: people sharing some idea, activity, or fandom tend to connect so that the concentration of edges in a given graph area increases. The (informal) definition of a *community* is similar to that of a cluster in data: *community members are more strongly connected than non-members*.

To formalize this notion, we will proceed as follows: let us consider that every vertex v of a graph has a feature $c(v)$, taking its values among a closed set of values. We may ask the question: How strong is the tendency of vertices to get connected by edges to other vertices with the same feature value c? This phenomenon is called *assortativity*: if in a social network, Whovians connect more to Whovians and less to Walking Dead fans, and Walking Dead fans connect more to Walking Dead fans as to fans of any other serial, then the network may have a strong assortativity property for the serial fandom feature.

Before we give a formula for assortativity, we will define a very important notion called *modularity* of a graph.

Let us denote m, the number of edges of graph G, and $k(v_i)$, the degree of vertex v_i. We will start with the following question: Imagine we know the degrees of all vertices. Still, we don't know where edges are located (if a vertex has a nonzero degree, it necessarily has adjacent edges, but we don't know with which other vertices it is connected), so we can only guess. If all edges are equiprobable, what is the probability of having an edge between two vertices v_i and v_j? By definition, $k(v_i)$ edges leave v_i. The probability that one of their extremities is v_j equals the number of extremities of edges arriving at v_j, divided by the total number of edge extremities. The former is the degree of v_j, and the latter is $2m$ (since the graph has m edges and each edge has 2 extremities). As there are $k(v_i)$ edges having an extremity at v_i, we can deduce that the probability of having an edge between v_i and v_j is $k(v_i)k(v_j)/2m$.

How can we limit ourselves to assortative edges (edges connecting vertices with the same feature value c)? For that, we will use the *Kronecker symbol* $\delta_{c(v_i),c(v_j)}$ which is equal to 1 when $c(v_i) = c(v_j)$ and to 0 otherwise. Using this symbol, and knowing only the degrees and classes of vertices, the expected number of assortative edges is

$$\frac{1}{2}\sum_{v_i,v_j\in V}\frac{k(v_i)k(v_j)}{2m}\delta_{c(v_i),c(v_j)},$$

where the initial $\frac{1}{2}$ comes from the fact that the edge going from v_i to v_j and the one going from v_j to v_i are, in fact, the same, so that taking the sum, we count the edges twice. We will define *modularity* as being the average difference between the real behavior of the graph and the expected number of assortative edges, as calculated above:

$$Q = \frac{1}{2m}\sum_{v_i,v_j\in V}\left(A_{i,j} - \frac{k(v_i)k(v_j)}{2m}\right)\delta_{c(v_i),c(v_j)},$$

where $A_{i,j}$ is the adjacency matrix of the graph, and the initial factor $\frac{1}{2}$ is, once again, because when we take a sum over arbitrary v_i and v_j in V, all edges are counted twice. Modularity takes its values in the interval $[-\frac{1}{2}, 1]$.

Modularity has some bright sides, like not being impacted by isolated nodes and always providing a clustering of maximal modularity with connected clusters [19]. But despite these facts, it is definitely *not* an intuitive notion. Nevertheless, modularity is very important for detecting communities because it is easy to calculate and very sensitive to community formation. Whenever we change the class of a vertex so that this class increases the vertex's assortativity, the graph's modularity value will increase.

To illustrate this property of the modularity of a graph, we have taken, in Fig. 8.8, the same graph five times, changing the class of a single vertex at a time. The graph has nine vertices belonging to three classes (Red, Green, and Blue). It is immediately apparent to us humans that a maximum of assortativity can be attained by gathering vertices of the same color in each of the three triangles. The modularity value for the leftmost graph is -0.1570, and through our one-vertex-at-a-time class changes

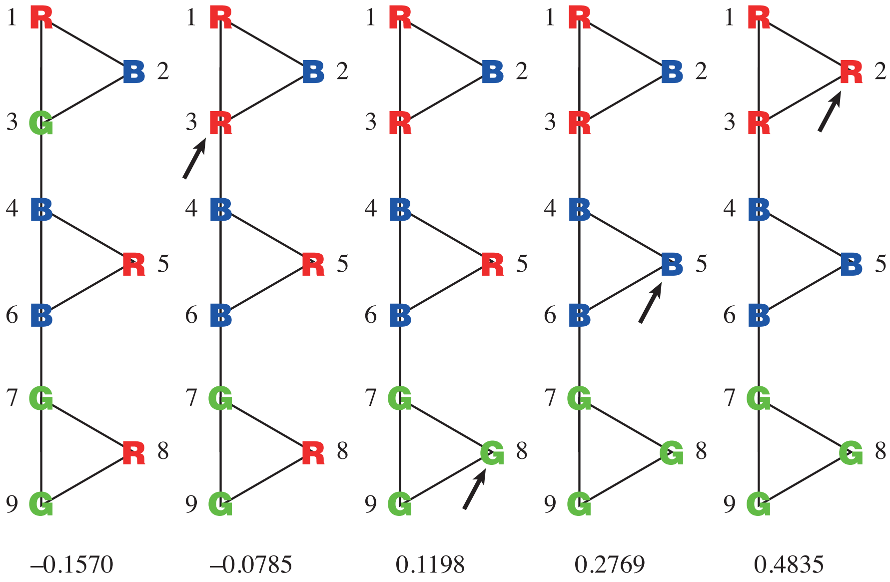


Fig. 8.8 A graph becoming more and more assortative and the corresponding modularity values

(indicated by an arrow), this value progressively moves to 0.4835. Each of the values *per se* is not important, but rather the fact that we are moving "in the right direction."

The code for calculating the modularity of the graph of Fig. 8.8 is the following:

```
1  import networkx as nx
2  import networkx.algorithms.community as nx_comm
3
4  G=nx.Graph()
5  for i in range(9):
6      G.add_node(i+1)
7  for v in [(1,2),(1,3),(2,3),(3,4),(4,5),(5,6),(4,6),(6,7),(7,8),(8,9),\
8      (7,9)]:
9      G.add_edge(v[0],v[1])
10 print(nx_comm.modularity(G,[{2,3,1},{4,6,5},{7,9,8}]))
```

The `i+1` on line 6 is needed because otherwise, we would get values between 0 and 8, while the vertices in the figure are numbered from 1 to 9. The second argument of the `modularity` function on line 9 is a list containing sets of vertices belonging to the three classes.

Among the many algorithms that do exactly this, namely assigning classes to vertices until maximal modularity is reached, there is one that is both fast, efficient for large graphs, and available in Python. This algorithm, called the *Louvain method* because it was developed by researchers during a stay at the Catholic University of Leuven ("Louvain"$_{\text{FR}}$ in French), uses a very impressive iterative method to reduce calculation: it works with passes, each pass consisting of a modularity maximization (as the one we did manually in Fig. 8.8) and a contraction. By contraction, we mean replacing an entire community with a single vertex. Depending on the graph's structure, it may take more or fewer passes, but contraction allows one to reduce the data size between passes drastically.

The Louvain method operates on edge-weighted graphs. The formula for modularity is still the same:

$$Q = \frac{1}{2m} \sum_{v_i, v_j \in V} \left(A_{i,j} - \frac{k(v_i)k(v_j)}{2m} \right) \delta_{c(v_i), c(v_j)},$$

but here $A_{i,j}$ is not the adjacency matrix, but the matrix of weights of edges and $k(v_i)$ is not the degree of vertex v_i but the sum of weights of edges attached to v_i.

In Python, the Louvain method has been implemented as a method of the *community* package:[8]

```
import community.community_louvain as nx_comm
partitions=nx_comm.best_partition(G,weight='weight')
```

where `G` is a graph, the edges of which have a weight called `weight`, and `partitions` is a dictionary with the vertex labels as keys and integers representing communities as values.

8.4.1 Two Examples Based on Shakespeare's *A Midsummer Night's Dream*

> Are you sure
> That we are awake? It seems to me
> That yet we sleep, we dream
>
> William Shakespeare, MND, IV.1

Shakespeare's *A Midsummer Night's Dream* (MND) is a vibrant play, combining the real and fictional worlds without deaths, wars, and suffering, as in his tragic plays. It also inspired one of Woody Allen's best movies, *A Midsummer Night's Sex Comedy* (1982).

To better understand what will follow, here is a short plot summary of each Act:

> Act I: The play opens with Theseus, the Duke of Athens, planning his upcoming wedding to Hippolyta, the Queen of the Amazons. He receives a visit from Egeus, who is upset with his daughter Hermia's refusal to marry Demetrius, the man he has chosen for her. Hermia is in love with Lysander, and they plan to elope. Meanwhile, Helena, in love with Demetrius, follows him into the woods. This first group of characters, namely Theseus, Hippolyta, Egeus, Hermia, Lysander, and Demetrius, are (human) nobles.
>
> The fairy king and queen, Oberon and Titania, are quarreling in the forest. Oberon orders his fairy servant Puck to find a magical flower whose juice can make anyone fall in love with the first person he/she sees so that he can use it on Titania. Puck accidentally uses the

[8] You will need to pip-install *community* but also the *python-louvain* package.

flower on Lysander, causing him to fall in love with Helena instead of Hermia. The second group of characters, namely Oberon, Titania, and Puck, are fairies.

Act II: The mechanicals, a group of craftsmen, gather in the woods to rehearse a play they hope to perform at Theseus and Hippolyta's wedding. Puck transforms Bottom, one of the mechanicals, into a donkey, causing the other mechanicals to run away in terror. Titania awakens and falls in love with the donkey-head Bottom (in the words of Shakespeare: "Titania waked, and straightway loved an ass."). We have here a chaotic interaction between fairies and humans, and two characters (a human, Lysander, and the queen of the fairies, Titania) falling in love with the wrong persons.

Act III: The four (human) lovers, Hermia, Lysander, Demetrius, and Helena, become entangled in a confusing web of love and jealousy in the forest. Puck tries to remedy the situation by using the magical flower, but again, the wrong persons fall in love (Lysander and Demetrius with Helena), causing even more chaos. Oberon realizes his mistake and orders Puck to set things right as soon as they all wake up. He also reconciles with Titania.

Act IV: The mechanicals reunite and perform their play, "Pyramus and Thisbe," at Theseus and Hippolyta's wedding. The play amuses everybody. Afterward, the lovers and fairy king and queen reconcile, and Puck delivers an epilogue.

Act V: The play ends with Theseus and Hippolyta, the lovers, and the fairy court sleeping and presumably dreaming. Puck delivers a final soliloquy, suggesting that the play was a dream.

To summarize, we have four groups of people: noble humans (Theseus, etc.), fairies, "mechanicals" (Quince, Snug, Bottom, etc.), and characters of the play in the play (Pyramus, Thisbe, etc.). We keep this configuration, even though in the play, the mechanicals are playing the characters of the play in the play (Bottom plays Pyramus, Flute plays Thisbe, etc.). As Shakespeare's troupe and budget were not unlimited, some of his actors played more than one role in the same play. According to Weiner [20, p. 339], the same actors played the secondary fairy roles and the craftsmen, so these characters never appear simultaneously on stage. This is something that the graph will confirm.

8.4.1.1 The Co-presence on Stage Graph

We will use graph theory to get insight into the importance of the various characters of the play, according to their co-presence on stage. For this, we have fetched the text from the Gutenberg Project and wrote down the entries and exits of the characters in the nine scenes of the play. Here is the format we used:

```
+The,Hip,Phi
-Phi
+Ege,Her,Lys,Dem
-The,Hip,Ege,Dem
+Hel
-Her
...
```

meaning that in Act I, Scene 1, Theseus, Hippolyta, and Philostrate enter together, then Philostrate exits, then Egeus, Hermia, Lysander, and Demetrius enter together, then Theseus, Hippolyta, Egeus and Demetrius exit together, Helena enters, and, finally, Hermia exits. This file has 90 lines, out of which 47 correspond to entries.

We took a very simplistic approach: whenever one or more characters entered, we incremented the weight of edges between them and all characters already present on stage. This is a simplification because, at some point, some characters are asleep, so they are not actively participating in the play. We chose to use an undirected graph since co-presence on stage is symmetric.

We obtained a graph with 27 vertices (the 27 named characters of the play) and 172 edges, which is, in fact, extremely sparse. Indeed if all characters were simultaneously on stage, we would have a complete graph K_{27} with not 184 but $27! = 1{,}09 \times 10^{26}$ edges.

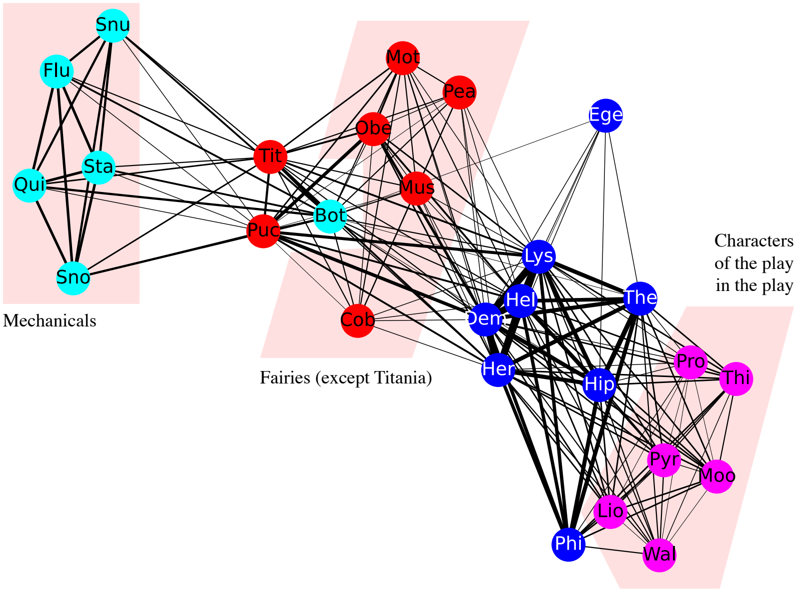


Fig. 8.9 Representation of the co-presence on stage graph of the characters of *A Midsummer Night's Dream. The color codes are as follows: blue for nobles, red for fairies, cyan for mechanicals and magenta for characters of the play in the play.*

The reader can see a representation of the graph in Fig. 8.9. In this graph, communities have been calculated by the Fruchterman-Reingold algorithm, with the `spring_layout` method [8] and given distinct colors (magenta, red, dark blue, and cyan). The width of strokes is proportional to the frequency of co-presence on stage. We recognize the nobles (in blue), and if we look carefully among the blue nodes, we see that the "infernal quartet" (Lysander, Hermia, Demetrius, and Helena) are very closely connected.

On the lower right corner, in magenta, we find the actors of the play in the play. In the play, they are indeed the same characters as the mechanicals (upper left corner, in cyan) and historically were played by the same actors as the fairies (red group in the middle). In the figure, we have used a pink background to highlight three groups of characters: mechanicals on the left, fairies (except Titania) in the middle, and characters of the play in the play on the right. The reader can check that no edge connects these groups so that the same actors can play them, provided they have enough time to change their costumes. This is evidence in favor of Weiner's hypothesis [20, p. 201]. Among the mechanicals, Bottom is an exception: although belonging to the group of mechanicals, he is graphically placed among the fairies since he is the one the (bewitched) Titania, queen of the fairies, falls in love with. Notice that all edges between the five craftsmen on the left and the fairies connect the former to Titania and Puck. Last observation: Egeus is an outlier, appearing only in specific scenes. He is connected to the nobles and Bottom since in Act IV, scene 1, when he discovers the sleeping characters. Among them is Bottom, who wakes up last after everybody else has left.

8.4.1.2 Doubling in MND

In the history of theater, the term "doubling" denotes an actor playing two or more roles in the same production. Shakespearian doubling has been extensively studied. Brett Gamboa has dedicated a book to this topic: *Shakespeare's Double Plays* [9]. On p. 139 he gives the following list of doubled characters for MND, to which we have added (in italics) the correspondence between Mechanicals and Thisbe actors, as given in MND:

Oberon = Theseus
Titania = Hippolyta
Puck = Philostrate
Lysander
Demetrius
Bottom = *Pyramus*
Hermia = Snug = Moth = *Lion*
Helena = Flute = *Thisbe*
Starveling = Cobweb = *Moonshine*
Snout = Mustardseed = *Wall*
Egeus = Peaseblossom = Quince = *Prologue*

We took the co-presence graph and applied a merge of nodes according to the correspondence between Mechanical and Thisbe actors. After this merge, our graph has 21 nodes and 151 edges (instead of the 172 edges we had previously). The problem of co-presence on stage is similar to the problem of *coloring*. In this problem, we search for the minimal number of vertex colors so that an edge always connects vertices of different colors. This is called the *chromatic number* of the graph. The algorithm is simple: after defining an order of colors, one takes vertices individually and assigns them the smallest color different from their neighbors' colors. As this depends on the order by which the vertices are considered, one runs the algorithm repeatedly (we ran it 100,000 times) with randomly chosen vertex orders.

We did this and found that 13 actors are necessary to play all roles, as follows:

Oberon = Quince = Prologue
Titania = Philostrate
Puck = Theseus

Lysander
Demetrius
Bottom = Pyramus
Hermia
Helena
Starveling = Moth = Moonshine
Snout = Egeus = Cobweb = Wall
Hippolyta = Peaseblossom
Flute = Mustardseed = Thisbe
Snug = Lion

This list has some common points with the one by Gamboa: Lysander, Demetrius, and Bottom are undoubled in both lists. But our list misses the doubling of Hermia with Snug and Moth, Helena with Flute, etc. No matter how hard we tried, we couldn't do better than 13 actors.

Then we programmed Gamboa's list and ran a program to check if there is no co-presence of these characters on stage. What came out is that there are three edges in the graph that connect characters played by the same actor, according to Gamboa:

Hermia—Moth
Hermia—Snug (or Lion)
Helena—Flute (or Thisbe)

We looked into the play and found that in Act V, Scene I, l. 1860, Hermia enters (together with Lysander, Demetrius, and Helena), and then, while there is no indication of Hermia leaving, at l. 1970, Lion enters together with Pyramus, Thisbe, Wall, and Moonshine. Is there an error? Or does Hermia sneak off stage and reappear as a lion? It would be interesting to find out.

In any case, the chromatic number algorithm allowed us to find a configuration of actor doubling compatible with our script version. As for Gamboa's configuration, only three co-presence edges contradict it, out of 151, involving Hermia and Helena.

8.4.1.3 Centralities of MND Characters in the Co-presence on Stage Graph

We applied the various centrality measures to this graph and obtained the following results.

In Fig. 8.10, the reader can see the eigenvector centrality of the characters. The four most central characters are the two couples: Lysander/Hermia, Demetrius/Helena. These four characters have exactly the same centrality values. Next comes Bottom, who plays an important role not because of his intellect or human qualities but because Puck transforms him into a donkey, and Titania falls in love with him. He, therefore, interacts both with humans and with fairies. Next come the actors of the play in the play, and after them, the fairies. Titania and Puck are slightly more central than the other fairies. After all, they play leading roles. Follows Egeus as an outlier, and finally, the least central characters are the mechanicals, who do not interact with any other character (except Titania and Puck). This can be seen in Fig. 8.9.

In Fig. 8.11, the reader can see the harmonic closeness centrality of the characters. As the distance between the vertices, we used the inverse of the frequency of

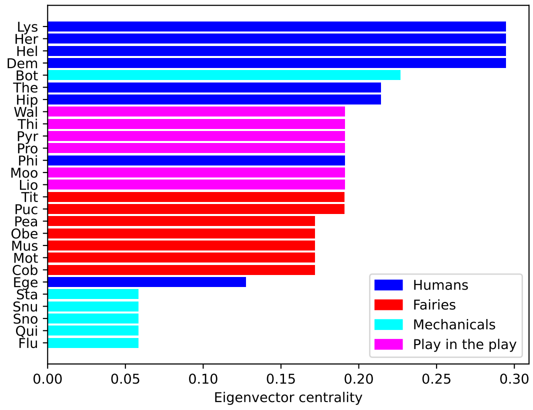


Fig. 8.10 Eigenvector centrality of the characters of MDN

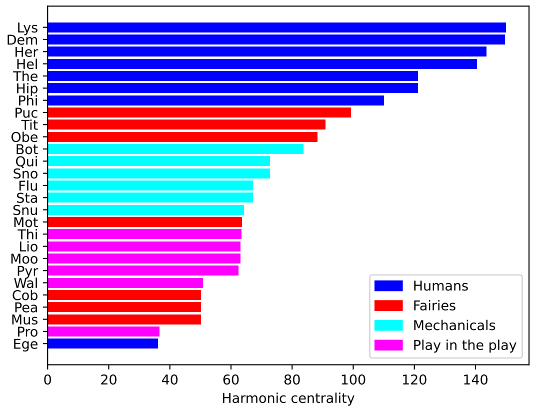


Fig. 8.11 Harmonic closeness centrality of the characters of MDN

co-presence on stage. The results are pretty different from those of the previous measure. The four protagonists are still highest-ranked, but we can detect minor differences: Lysander and Demetrius are more central than Hermia and Helena (so much for Shakespeare's gender correctness). Then come the three other nobles, followed by Puck, Titania and Oberon. In the middle of the ranking scale, we find the group of mechanicals. The reader may be surprised because, in Fig. 8.9, these vertices are displayed at the extreme left part of the graph, contradicting the geometric centrality this measure should evaluate. The problem lies in the rendering of the graph: centrality measure values are unique and factual, but there are infinite ways to display a graph. The various renderings can trigger intuitions but do not prove anything.

In Fig. 8.12, the reader can see the betweenness centrality of the characters. As distance between the vertices, we again used the inverse of the frequency of co-presence on stage. The results are impressive: most characters have zero betweenness centrality. The highest ranked are Titania and Bottom. The "affair" between

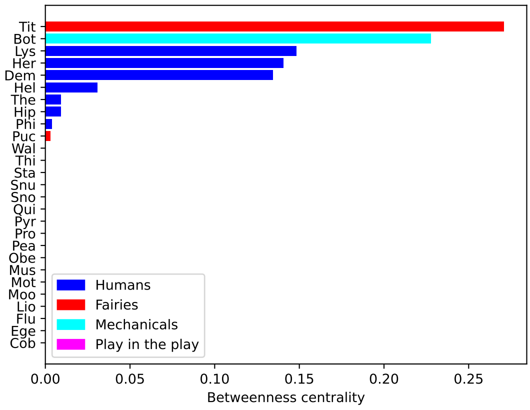


Fig. 8.12 Betweenness centrality of the characters of MDN

them turned them into "brokers," i.e., vertices through which one has a higher probability of passing when going from one side of the graph to the other, for example, from fairies to craftsmen. Then we see the infernal lovers, and again, there is a difference between them: Lysander and Hermia are higher ranked, then comes Demetrius, and Helena is much lower ranked. This is probably due to Puck: he drops the liquor on Lysander's and Demetrius's eyes and therefore has strong links with them and not with Helena, as can be seen by observing the widths of the Puck-Lysander and Puck-Demetrius edges in Fig. 8.9. In contrast, the Puck-Helena edge remains thin.

As we saw, the three centrality measures gave quite distinct results, allowing us to get insight into the interaction between the play's characters. Let us now apply the Louvain method to obtain communities of characters.

8.4.1.4 Communities of MND Characters in the Co-presence on Stage Graph

We applied the Louvain community detection method and obtained the following four communities:

1. Bottom, Egeus, Helena, Hippolyta, Theseus;
2. Lion, Moonshine, Philostrate, Prologue, Pyramus, Thisbe, Wall;
3. Flute, Puck, Quince, Snout, Snug, Starveling, Titania;
4. Cobweb, Demetrius, Hermia, Lysander, Moth, Mustardseed, Oberon, Peaseblossom.

The second community is one of the actors in the play, plus Philostrate, who appears only at the end to inform Theseus about the play and, therefore, is connected to it. The third community comprises mechanicals (except Bottom), plus Puck and Titania. The choice of affecting Puck to this category is surprising since he cooccurs with fairies and lovers (mostly in category 4). Also, Titania is neither in the same community as Oberon nor in the one of her beloved donkey-headed Booth. The fourth

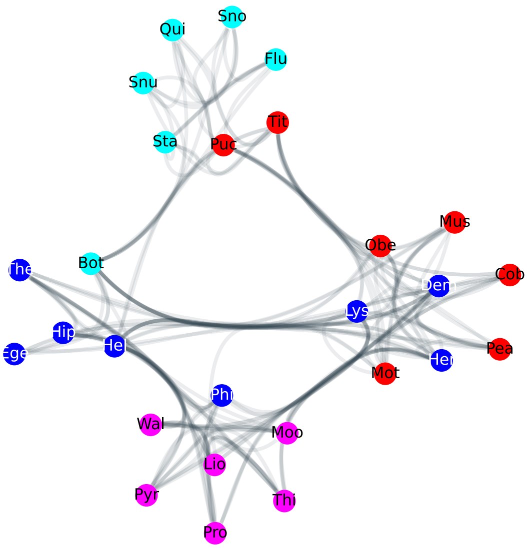


Fig. 8.13 Communities of the characters of MDN, obtained through the Louvain method

category gathers all remaining fairies and three of the young protagonists, namely Lysander, Hermia, and Demetrius, but not Helena—for some reason, 428 years after Shakespeare, fate (in the guise of the method of Leuven) still does not want Helena to join Demetrius. More seriously, how can we explain this counter-intuitive decision of the Louvain method to place Helena among Bottom, Egeus, Hippolyta, and Theseus? To find a precise answer, one must observe all calculations and all intermediary steps of the Louvain contraction process. Maybe this is due to the optimization process: the Louvain method seeks groups of vertices with maximum modularity, and it probably has happened that the modularity of community #4 was less with Helena than without her, so she has been "sacrificed for the common good," i.e., moved to community #1, the modularity of which she increased.

Figure 8.13 displays the corresponding intra- and inter-community links of the four communities. It has been produced by the Python package *netgraph*.

8.4.1.5 The Vocative Graph

One can argue that studying the co-presence on stage is not really an NLP problem. Our following example will look at addressing or mentioning somebody by his/her name, whether present or absent. The nodes will be individuals (characters of the play or other mythological named entities), and we will follow a simplistic approach: we will draw an edge whenever a character addresses a person. The weight of the edge will be the number of times the person has been addressed. For that, we will use *stanza* to obtain all named entities of type "Person." After some filtering, we use a Python script that keeps track of the speaker and, on detecting a named entity in the list (an “addressee”), will increment the weight of the addresser-addressee edge. We will call this graph the *vocative* graph of MDN.

The reader can see the resulting graph in Fig. 8.14. This time, it is a digraph, since addressing is not symmetric, we have an addresser and an addressee. We again used Fruchterman-Reingold’s algorithm, which placed the external referents (entities that are not characters of the play) in the periphery. Most of them are hapaxes: they are addressed/mentioned only once. Peripheral vertices have one thing in common: they have only incoming edges. They are addressed but do not address or mention anyone.

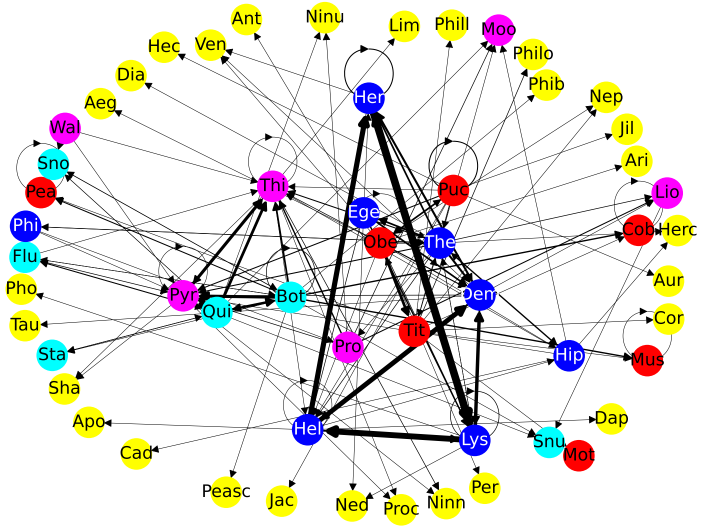


Fig. 8.14 Representation of the vocative graph of MDN. The color codes are the same as for the co-presence on stage graph, plus yellow for individuals who are not characters.

Edge width represents the frequency of addressing: the absolute champions are Lysander and Hermia, addressing each other all the time, and this is what we would expect from people in love, even in the 16th century. Helena forms a triangle with Hermia and Lysander and another with Lysander and Demetrius. We also see many loops. They represent the case where characters address or mention themselves, as in Puck’s epilog (Act V):

> And, as I am an honest Puck,
> If we have unearned luck
> […]
> Else the Puck a liar call.
> So, good night unto you all.

The graph of Fig. 8.14 is also quite efficient in highlighting the characters that engage in verbal exchange. They are in the middle of the graph. One sees immediately that among the mechanicals, Bottom is very prolix, and so is Quince. All others play secondary roles.

8.4.1.6 Centralities of MND Characters in the Vocative Graph

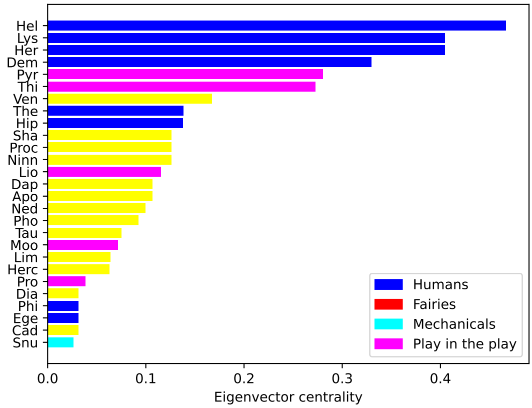


Fig. 8.15 Eigenvector centrality of the vocative graph of MDN

We calculated the eigenvector centrality of the vocative graph. The reader can see the results in Fig. 8.15. The four lovers are again on top, with Helena being the most central, followed by Lysander, Hermia (ex aequo), and Demetrius. The next characters in the order of decreasing centrality may appear unexpected: Pyramus and Thisbe, the protagonists of the play in the play, followed by goddess Venus, mentioned only three times in the whole play. How can Venus be more important than, e.g., Puck, a figure that appears throughout the play? Has the algorithm fallen under the spell of the goddess of beauty? This is a typical side-effect of the ingoing eigenvector approach: centrality is inherited from the predecessors. Venus is mentioned only thrice but by at least two very central characters, namely Hermia and Demetrius. Therefore, Venus inherits from their centrality. Puck is invisible to humans. None of the high-ranked lovers ever addresses or mentions him, so neither his name nor any other fairy appears in the list of central characters.

The harmonic centrality is not very different from the eigenvector one; therefore, we will not display it. We do display the betweenness centrality in Fig. 8.16. Helena is again first, but the real surprise is Thisbe, which is second in the ranking: indeed, if we observe the graph, Thisbe has a high ingoing degree, and therefore many paths go through her vertex. One may ask: Hermia and Lysander have much higher incoming degrees. Why aren't they higher ranked than Thisbe? The answer is that betweenness is relative: the number of paths going through a given vertex divided by the total number of paths. In the case of Hermia and Lysander, many paths are going through them, but also, many paths are taking alternative routes, and the ratio happens to be

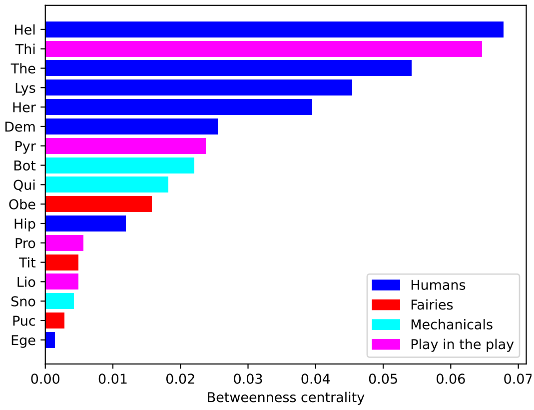


Fig. 8.16 Betweenness centrality of the vocative graph of MDN

lower than the one for Thisbe. Otherwise, as expected, no external characters can have nonzero betweenness centrality since they have no outgoing edges (being not there, they cannot address anyone).

8.4.1.7 Communities of MND Characters in the Vocative Graph

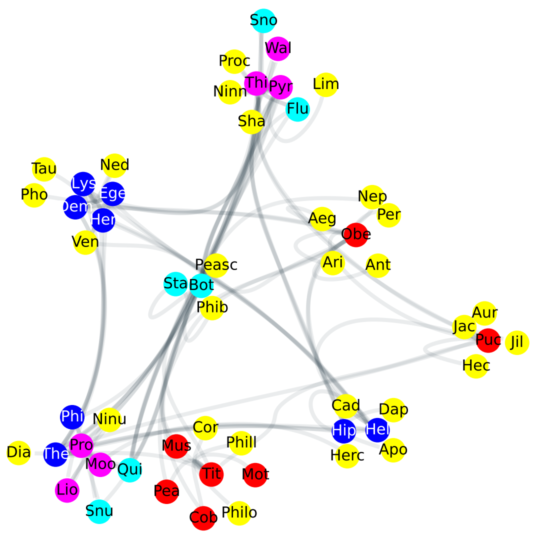


Fig. 8.17 Communities of the vocative graph of MDN, obtained through the Louvain method

We also used the Louvain method to obtain communities in the vocative graph. The result is displayed in Fig. 8.17. Eight classes are identified. Unsurprisingly, each class contains some characters of the play, surrounded by external individuals. Oberon and Puck have their classes. All other fairies are together in a single class. Play-in-the-play characters are equally divided into two classes: one which contains Prologue, Moonshine, Lion and also Philostrate, Theseus, Quince, and Snug; and one that contains Wall, Thisbe, and Pyramus, and also Snout and Flute. What is astonishing is that in both cases, the actor and role are in the same community, since Quince plays the Prologue, Snug plays the Lion, Snout plays the Wall, and Flute plays Thisbe. Finally, once again, the three lovers, Hermia, Lysander, and Demetrius, are in the same community, and Helena is in a different community with Hippolyta.

8.4.1.8 Possible Improvements

We have used various graph techniques on data extracted from Shakespeare's MND. The approach is very simplistic: instead of separating addressing and mentioning, we have just collected named entities and used them without considering their function. Also, we omitted indirect addresses. For example, in Oberon's lines (Act II, Scene 1):

> How canst thou thus, for shame, Titania,
> Glance at my credit with Hippolyta,
> Knowing I know thy love to Theseus?
> Didst not thou lead him through the glimmering night

Oberon addresses Titania in person, mentions Hippolyta and Theseus, and then again addresses Hippolyta indirectly ("thy," "thou") and mentions Theseus indirectly ("him"). Coreference detection would be needed here to better understand addresses and mentions.

One could apply this approach to Woody Allen's *A Midsummer Night's Sex Comedy*. In MND there are four lovers: Hermia and Lysander love each other, Demetrius wants to marry Hermia, and Helena is Demetrius's ex-girlfriend and is still in love with him. In Woody Allen's movie, there are not four, but six "lovers": Leopold wants to marry Ariel, Maxwell is married to Adrian but is secretly in love with Ariel with whom he had a (platonic) relation in his youth, Maxwell has had sex with Adrian but falls in love with Ariel, and Leopold dies while having sex with Dulcy. One could say that Ariel is inspired by Hermia, Leopold by Demetrius, Dulcy by Helena, and both Andrew and Maxwell by Lysander. The graph of feelings and relationships is more complex, but the co-presence on stage graph and the vocative graph are simpler since there are no other characters in the movie but these six.

8.5 Further Reading

8.5.1 Literature

Textbooks on graph theory differ in the amount of mathematics and/or computer code they provide. Newman [17] is an excellent textbook with formulas but without theorems and their proofs, and without code. The author believes the best book in the universe for the reader interested in graph algorithms is CLRS, aka Cormen, Leiserson, Rivest & Stein [6]. Chapter 6 is dedicated to graphs. Leiserson's lectures on Youtube[9] are excellent. For the reader interested in the mathematical theory of graphs, it is unfolded very reader-friendly in Diestel [7]. The French-speaking reader should consult 's books, such as [2]. They are eternal masterworks of clarity and elegance.

Besides being a pioneer in graph theory, Berge was also a member of the Oulipo[10] group and as such, published a short story where a crime is solved by graph theory: *Who killed the Duke of Densmore?* ([3], and English translation [4]). Graphs are used to model the co-presence of the various ex-wives of the victim, and the solution is given by the mathematical impossibility of a specific ex-wife's testimony so that only she can be the murderer.

Mihalcea [15, 16] are dedicated to graphs applied to Natural Language Processing. He discusses the use of graphs in semantics and syntax, and describes applications such as summarization, keyword extraction, topic identification, machine translation, and question-answering.

8.5.2 LaTeX

To draw graphs, use the *tikz* package, a very powerful package with very rich documentation.

---

[9] `https://www.youtube.com/playlist?list=PLNfr0R3crNvIWLZM0qsVAHaO-U9r1XE2q`, the part on graphs starting at Lecture 16.

[10] The *Oulipo* "Ouvroir de litérature potentielle"$_{\text{FR}}$ ⌊Workshop of potential literature⌋ is a group of writers and mathematicians founded in 1960 by the famous author Raymond Queneau (known for "Zazie dans le métro" and "Exercices de style" and the science historian François Le Lionnais. It had notable members such as Georges Perec, who wrote an entire novel without the letter "e" (*La disparition*) and a second novel with "e" as the only vowel (*Les revenentes*), and Italo Calvino who wrote the novel *If on a winter's night a traveler* with eleven unfinished subnovels. Oulipo cultivated constrained writing, where one or more authors set themselves a formal constraint and wrote texts under it. A famous example is the "Our Father who art in heaven" prayer written using solely Paris subway station names. It begins, "Notre Auber qui êtes Jussieu, Que Simplon soit Parmentier, Que Ta Volontaires soit Place des Fêtes…" (`http://classes.bnf.fr/ecrirelaville/ressources/le_tellier.pdf`, p. 6).

8.5.3 Science Fiction

Egon's *Schild's Ladder* (2002) involves quantum graph theory. An experiment that was supposed to create a new kind of vacuum that would exist only for a few nanoseconds went wrong. The "novovacuum" expands at half the speed of light, swallowing inhabited solar systems. Scientists want to destroy it and discover it carries life. As a reviewer suggested, this novel is a variant of Frankenstein on a stellar scale. The incipit of the book is a graph-theoretic variant of the incipit of the book of Genesis:

> In the beginning was a graph, more like diamond than graphite. Every node in this graph was tetravalent: connected by four edges to four other nodes. By a count of edges, the shortest path from any node back to itself was a loop six edges long. Every node belonged to twenty-four such loops, as well as forty-eight loops eight edges long, and four hundred and eighty that were ten edges long. The edges had no length or shape, the nodes no position; the graph consisted only of the fact that some nodes were connected to others. This pattern of connections, repeated endlessly, was all there was.

8.6 Exercises

Exercise 7-1: Using WordNet for disambiguation

With the help of ChatGPT, we have prepared a corpus of 100 sentences using the word "bank." Fifty among them refers to the meaning of financial institution, and the other fifty to the meaning of terrain alongside a river.

Some sentences are unambiguous:

> The bank collaborates with local charities to support community initiatives.
> We spotted a beaver dam along the bank.
> The bank's staff is always polite and helpful.
> Ducks and geese often gather at the bank.

Others are ambiguous, either because a bank (financial institution) is also a building that can include natural elements or lend itself to activities other than finance or because a river bank can be guarded if it happens to be a state border, etc.,

> The old willow tree leans over the bank.
> The bank's security measures include surveillance cameras and alarm systems.

Use the graph structure of WordNet to disambiguate these sentences.

Exercise 7-2: Find the most central word of the Quran and the Bible

Parse English Quran and Bible translations with a dependency syntax parser. For each book, build a graph of nouns/proper nouns connected by an edge whenever they have dependencies with the same verb in a sentence. Take the largest connected component and find the twenty most central nouns/proper nouns for the closeness centrality.

Exercise 7-3: Assortativity of Chinese Hyperonyms

Sinograms are made of components so that we can consider a word as an ordered sequence of components (the way components are assembled is important, but nevertheless, 57.5% of Unicode's sinograms are of the form AB where A and B are combined horizontally). Do we keep some components when we move from a word to one of its hyperonyms? From a graph-theoretical point of view, if words are vertices, components are features of vertices, and edges are hyperonym relations between words, the question asked is one of the assortativity of the graph with respect to these features.

Calculate the assortativity of the graph of Chinese WordNet. In a second calculation, consider only the right parts of AB sinograms.

Exercise 7-4: Productivity of sinographic component base

Sinograms can be decomposed into components (see Exercise 7-3). We call a sinogram *primitive* if it cannot be decomposed into components (for a given decomposition framework) and *decomposable* if it can be decomposed into at least two primitives. Let D be a set of decomposable sinograms and P a set of primitive sinograms. We will say that P is the *base set* of D, and D the *production set* of B when

1. all elements of D can be decomposed into elements of P and only into elements of P;
2. the elements of D are the only elements generated by the elements of P.

We will consider that P is a subset of the Unicode table of "CJK Unified Ideographs" (0x4E00–0x40BF).

Find the largest base and production sets in the following cases:

- D is the Unicode table "CJK Unified Ideographs" (0x4E00–0x40BF);
- D is the merge of the Unicode tables "CJK Unified Ideographs" and "CJK Unified Ideographs Extension A" (0x3400-0x4DBF);
- D is the merge of the Unicode tables "CJK Unified Ideographs," "CJK Unified Ideographs Extension A" and "CJK Unified Ideographs Extension B" (0x20000-0x2A6DF);
- D is the Jōyō kanji set used in Japanese education for all grades;
- D is the Jōyō kanji set used in Japanese education for each grade, cumulatively.

Knowledge of these sets can help teach sinograms or for memorizing them.

References

[1] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: *International AAAI Conference on Weblogs and Social Media*. 2009.

[2] Claude Berge. *Graphes*. 3rd ed. Gauthier-Villars, 1983.

[3] Claude Berge. *Qui a tué le duc de Densmore?* Vol. 67. La bibliothèque oulipienne. Oulipo.

[4] Claude Berge. "Who killed the Duke of Densmore?" In: *Oulipo Laboratory*. Alas Anticlassics, 1995.

[5] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN System* 30 (1998), pp. 107–117.

[6] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd ed. The MIT Press, 2009.

[7] Reinhard Diestel. *Graph Theory*. 5th ed. Vol. 173. Graduate Texts in Mathematics. 2017.

[8] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164.

[9] Brett Gamboa. *Shakespeare's Double Plays*. Cambridge University Press, 2018.

[10] Martin Gardner. "Word Ladders". In: *The Edge of the Universe: Celebrating Ten Years of Math Horizons*. Ed. by Deanna Haunsperger and Stephen Kennedy. Mathematical Association of America, 2006, pp. 22–23.

[11] David F. Gleich. "PageRank beyond the Web". In: *SIAM Review* 57 (2015), pp. 321–363.

[12] Yifan Hu. "Efficient and High Quality Force-Directed Graph Drawing". In: *Mathematica Journal* 10 (2005), pp. 37–71.

[13] Leo Katz. "A New Status Index Derived from Sociometric Analysis". In: *Psychometrika* (1953), pp. 39–43.

[14] Donald E. Knuth. *Selected Papers on Fun and Games*. University of Chicago Press, 2011.

[15] Rada Mihalcea and Dragomir Radev. *Graph-Based Natural Language Processing and Information Retrieval*. Cambridge University Press, 2011.

[16] Vivi Nastase, Rada Mihalcea, and Dragomir R. Radev. "A survey of graphs in natural language processing". In: *Natural Language Engineering* 21 (2015), pp. 665–698.

[17] Mark Newman. *Networks. An Introduction*. 2nd ed. Oxford University Press, 2018.

[18] Larry Page. "The PageRank Citation Ranking: Bringing Order to the Web". `cis.upenn.edu/~mkearns/teaching/NetworkedLife/pagerank.pdf`. 1998.

[19] Daniel Delling Ulrik Brandes and et al. "On Modularity Clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 20 (2008).

[20] Andrew D. Weiner. ""Multiformitie Uniforme": A Midsummer Night's Dream". In: *English Literary History (ELH)* 38 (1971), pp. 329–349.

Chapter 9
Formal Languages

> […] what is so special about formal languages and formalisms more generally? Whence the magic?
>
> C.D. Novaes, *Formal Languages in Logic*

9.1 Background

As underlined by Dutilh Novaes [19, pp. 11–28], when we refer to formal languages, *formality* corresponds to mainly two properties: *asemanticity* (in the sense that the objects we are dealing with have no specific meaning, and are just abstract entities) and *calculation* (the purpose of these languages is to allow calculations, in particular by computers).

In this book, we will consider that formal languages are mathematical objects used to model situations in the real world. More precisely, formal languages are used to model *certain* situations in the real world. This particular kind can be described as follows: the objects to model have to be discrete, and there has to be a way of "concatenating" these objects to build *sequences* of objects. At first, these conditions seem very restrictive. Take a simple natural phenomenon such as ambient temperature: it cannot be modeled by a formal language since it is not discrete (temperature, time, and space are continuous), and "concatenating temperatures" does not make sense. But continuous phenomena can be discretized, and concatenation can represent the relation "followed by": if we denote temperature by integer degrees °C and discretize time by hourly intervals and space by assuming that an entire town has the same (average) temperature, then 12-15-18-21 is a word of a formal language consisting of temperatures taken at hourly intervals. Formal languages are applied in many domains other than natural language processing and linguistics: biology (DNA strains), programming, compression, etc.

To return to this book's frame, we will mainly use formal languages to model natural languages. Before we start, let us evoke a problem of terminological nature: the building blocks of formal languages are "words" and "letters," but when modeling natural languages, formal language words will not stand for natural language

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_9

words, as expected, but also for other units, for example, phrases, sentences, text blocks, morphemes, graphemes… For example, when we study constituent syntax, ☞ 97 we consider constituents (which are sequences of natural language words) as formal language *words*. There is no problem in doing so. It is only the terminology that may be confusing. We will call "formal letters" and "formal words" the building blocks of formal languages whenever necessary.

9.2 Basic Definitions

Let Σ be an arbitrary discrete set, which we will call a (formal) *alphabet* for purely historical and eurocentric reasons. A (formal) *word* of length n over Σ is a map from $\{1, \ldots, n\}$ to Σ, i.e., a choice of elements of Σ which will be at position 1, position 2, …, position n of the word. A formal language over Σ is a (potentially infinite) set of words over Σ, including the empty word ϵ.

Instead of using a map from $\{1, \ldots, n\}$ to Σ, we could say that (formal) letters are (formal) words of length 1, also called *singletons*, and that there exists a *concatenation operator* sending two words of length p and q to a word of length $p + q$ (for any p and q). There is no specific symbol for concatenation: we write the two words side by side: $abcd$ is the result of the concatenation of letters a, b, c and d, or of the concatenation of words ab and cd, etc.

Let us denote by Σ^* the set of all words over Σ plus the empty word ε, i.e., the largest possible formal language over Σ. The set Σ^* has an algebraic structure: let us take two words $w, w' \in \Sigma^*$, their concatenation is again in Σ^*, i.e., concatenation is an algebraic operation of Σ^*. And since we have $\epsilon \cdot w = w \cdot \epsilon = w$, we see that ϵ is the neutral element of Σ^* for the concatenation operation. Readers acquainted with the usual algebraic structures (groups, rings, fields, etc.) will be disappointed: Σ^* is none of these structures. What we miss is the invertibility property: if you have a word abc there is nothing you can concatenate it with to obtain the empty word ϵ. Cayley has called such an algebraic structure a *monoid* and Σ^* is the *free monoid* over Σ. The name may sound strange, but there is a monoid we all know and respect: $(\mathbb{N}_0, +)$, the set of natural numbers including zero, with the addition operation. This set that some mathematicians claimed to be of divine origin,[1] is, in fact, a monoid.

In our context, we will not even have the monoid property: σ^* is a monoid, but an arbitrary formal language is just a subset of a Σ^*, and thus not *necessarily* a monoid. Indeed, if ab and cd belong to a language L, $abcd$ may not belong to L.

Great care should be taken to avoid confusion: when we write "$aaaa$," we mean $a_1a_2a_3a_4$, where a_1, a_2, a_3 and a_4 are different instances of the letter $a \in \Sigma$. It is simpler to avoid indexing instances of letters, but to emphasize that they are different instances, we will never write a^4 as an abbreviation of $aaaa$, as in algebra. If necessary, a word consisting of n instances of letter a will be written

[1] Kronecker claimed, at a meeting of the Berlin Assembly of German Naturalists and Physicians, in 1886, that "God created natural numbers, and everything else is the work of (wo)men."

$$\underbrace{a \cdots a}_{n \text{ times}}.$$

Similarly, one should not confuse the (formal) letter $a \in \Sigma$ with the singleton $a \in \Sigma^*$.

Having given the basic definition of a formal language, let us move higher and consider the set of formal languages over the same alphabet Σ. As a formal language is simply a subset of Σ^*, the set of all formal languages is the set of all subsets of Σ^*, which we denote by $\wp(\Sigma^*)$. Interestingly, Σ^* is of discrete cardinality (cardinality $\aleph_0$) but $\wp(\Sigma^*)$ is of continuous cardinality (cardinality $\aleph_1$) similarly to $\mathbb{Q}$ (discrete) and $\mathbb{R}$ (continuous).

There are some very useful operations in $\wp(\Sigma^*)$: the *complement* $\bar{L}$ of a language L (all words of Σ^* not belonging to L), the *concatenation* $L_1 \cdot L_2$ of two languages (consisting of words $w_1 w_2$, where $w_1 \in L_1$ and $w_2 \in L_2$), the (set-theoretical) union and intersection of languages, and the free monoid L^* over a language (consisting of all concatenations of words of L), also called *Kleene closure* of L.

9.3 Formal Grammars

In the previous section, we defined a formal language as "a subset" of Σ^*. Now comes the tricky part: how do we describe such a subset? If the language is finite, we can unambiguously define it by enumerating its words, as in $L = \{\epsilon, ab, abab\}$, this is called an *extensional description* of the language. But what if it is infinite (or very, very large)? What if we want to describe all possible grammatical sentences of a given natural language? Technically, this set is not infinite since we have neither infinite time to utter a sentence nor infinite space to write it down. Still, it is indeed very large in that, given any sentence, we can always produce a longer one by adding a few words.

To define formal languages, Chomsky introduced a tool called *formal grammars*. A *formal grammar over an alphabet* Σ is a quadruple (Σ, N, R, S) where N is a set of symbols not appearing in Σ, called *nonterminal symbols*, R is a set of *production rules*, and S an element of N called *starting symbol*. In this context, the elements of Σ are called *terminal symbols*.

A production rule is a quadruple (C_L, A, C_R, B) where $C_L \in (N \cup \Sigma)^*$ (C_L is called the "left context"), $A \in N$ (a nonterminal), $C_R \in (N \cup \Sigma)^*$ (C_R is called the "right context") and $B \in (N \cup \Sigma)^*$, where the * operator means the free monoid over a given set, including the empty word". A typical production rule is S $\longrightarrow$ NP VP, where S, NP, and VP are nonterminals, the left and right contexts are empty, A is S and B is the word concatenation of NP and VP (usually written "NP VP").

The usual notation for production rules is $C_L A C_R \longrightarrow B$. Production rules are also called *rewrite rules* because they allow to "rewrite words," in the sense that one takes A inside the left side of the production rule and replaces it by B, which is the right side of the production rule. For example, if we apply the rewrite rule $P \longrightarrow QR$ to the word $MNPST$ we obtain the word $MNQRST$. The reader should be aware of the fact that rewrite rules are *not* functions: if we apply the rewrite rule $P \rightarrow QR$ to the word $PPPP$ we can have, at our leisure, four different results: $QRPPP$, $PQRPP$, $PPQRP$, or $PPPQR$. To obtain $QRQRQRQR$, this rule has to be applied four times in a row. There are $4! = 24$ different ways of obtaining $QRQRQRQR$ out of $PPPP$.

Let $G = (\Sigma, N, R, S)$ be a formal grammar. When a word can be obtained from the starting symbol S after a finite number of applications of rewrite rules in R, we say that the word can be *derived* from the starting symbol. Chomsky defines *the language L defined by grammar* $G = (\Sigma, N, R, S)$ as the set of all words (containing only terminal symbols) that can be derived from S using rewrite rules from R.

So if we wish to find out whether a (formal) word belongs to a language L defined by a grammar G, we start deriving S until we obtain the word. If we need all words of a given length, we can apply all possible combinations of rewrite rules until the length is exceeded. A formal language can be generated by applying the rewrite rules
☞ 98 of a grammar, so Chomsky called these grammars, *generative grammars*.

9.3.1 The Chomsky Hierarchy

By restricting the type of rewrite rules, Chomsky defined a hierarchy of formal grammars and the corresponding languages.

Let us recall that an arbitrary rewrite rule is of the form $C_L A C_R \rightarrow B$, where A is a nonterminal, and $C_L, C_R, B \in (N \cup \Sigma)^*$. Going from the most general to the most specific kind of formal language, we say that

- a grammar is of *type 0* (or *unrestricted*) if there is no restriction on C_L, C_R, B;
- a grammar is of *type 1* (or *context-sensitive*) if B can be written as $C_L B' C_R$ (i.e., we have the same left and right contexts on both sides of the rewrite rule);
- it is of *type 2* (or *context-free*) if C_L and C_R are empty words (i.e., we have no context);
- it is of *type 3* (or *regular*) if C_L and C_R are empty and B is a terminal b or a terminal followed by a nonterminal bB'.

This distinction really makes sense only for infinite languages. When a language is finite, e.g., $L = \{w_1, \ldots, w_n\}$, we can always build as many rules $S \rightarrow w_i$ as their words, and such a grammar will always be regular. Nevertheless, this argument is of little use in real life, where we have the notion of a "very large set that could as well be infinite." A natural language is such a set. Chomsky argues that language is infinite since one can always add an additional word to a sentence simply by prefixing it with "X said that," not considering the lifespan of the utterer or even the entire human race. Replacing "infinite" with "very large," it makes sense to investigate
☞ 75 whether natural languages (other than Lojban) are regular, context-free, or context-sensitive. To make a long story short, Partee *et al.* [20] have shown that it is more efficient to consider English as a context-free language. There has been a long debate about whether context-sensitive languages exist. See Pullum's irreverential and sharp text [21, pp. 131–146]. Most tools used today for natural languages consider them to be context-free.

The following will discuss the simplest kinds of formal languages: regular and context-free.

9.4 Regular Languages

Regular languages behave very well: the union, intersection, Kleene closure, and complementary of a regular language is a regular language. Nevertheless, the reader should avoid the following confusion: saying that the *set of regular languages* is contained in, e.g., the *set of context-free languages* does not mean that we have the same relation of inclusion for languages as sets of words. As we will see in § 9.5, a regular language may contain a context-free non-regular language as a subset.

There are three tools for describing regular languages: *regular formal grammars*, *regular expressions*, and *finite state automata*. Context-free languages can be described by *context-free grammars* and by *pushdown automata*. When modeling the syntax of natural languages, we mainly use context-free languages. The Lojban language has been designed to be regular. ☞ 75

Regular expressions are among the most essential tools of a programmer, especially when processing linguistic data, converting data formats, cleaning up corpora, etc. Therefore, we will briefly describe them in the following section.

9.4.1 Regular Expressions

Mathematically speaking, a regular expression is the description of a formal language. Technically, it looks like a word to which we have added special symbols. Some authors call these symbols a *meta-alphabet*. Kleene first introduced regular expressions in 1951, in a RAND Corporation Report on events in nerve nets [14].

There are many syntaxes for regular expressions. In the following, we give the basic "abstract" syntax and a richer syntax used in most programming languages. The latter is also part of an IEEE standard, the POSIX syntax [12].

9.4.1.1 Abstract Syntax

The basic syntax of regular expressions uses parentheses, concatenation (denoted by $\cdot$), union ($\cup$), the star-operator (*), the empty word (ϵ)[2]. For example, if over the alphabet $\{a, b\}$ we wish to describe the language of words starting with ab, ending with ba and having an arbitrary central part, we will write

$$(a \cdot b \cdot ((a^*) \cup (b^*))^* \cdot b \cdot a).$$

[2] The empty word is denoted by λ in [25, p. 147].

This notation is used in textbooks, such as Simovici & Tenney [25]. Dehornoy [8, pp. 82–84] uses the notation

$$(\{a\} \cdot \{b\} \cdot (\{a\}^* \cup \{b\}^*) \cdot \{b\} \cdot \{a\}).$$

This emphasizes that letters need to be considered singletons to be concatenated.

9.4.1.2 POSIX Syntax

The syntax and the semantics of POSIX regular expressions are described in Chapter 9 of the POSIX.1-2017 standard [12]. As they are intended to be used in program-
☞ 343 ming languages, the alphabet Σ is fixed: it is the Unicode[3] encoding. Formal language words are character strings; formal language concatenation is simply character string concatenation. A choice between several characters is indicated by *bracket expressions*:

- a choice between `a`, `b` and `c` is written `[abc]`;
- using the codespace order of Unicode, a choice in a range of characters, for example, from `a` to `z`, is written `[a-z]`;
- the above features can be arbitrarily combined: a choice between an arbitrary Latin lowercase letter except "m" and "o" or an arbitrary Greek lowercase letter except "ζ," is written `[a-lnp-zα-εη-ω]`, i.e., all lowercase Latin letters between "a" and "l" (inclusive), letter "n," all lowercase Latin letters between "p" and "z," all lowercase Greek letters between "α" and "ε," and between "η" and "ω" (among the latter there are also the two versions of sigma, "σ" and "ς," since the interval of characters is the corresponding interval of Unicode positions);
- if the first character after the opening bracket is a hat `^`, the choice is not between characters given between brackets but between the complementary set of those characters: `[^0-9]` is any character that is *not* an Arabic digit. Notice that when taking complementary sets, the choice is quite wider: when writing `[a-z]`, one can choose between 26 characters. When writing `[^a-z]`, one has the choice between 288,807 (!) different characters (as of version 15 of the Unicode standard);
- to protect special characters (i.e., to search for them as if they were standard characters, without their special semantics), we precede them by a backslash: `\[`, `\]`, `\\`, etc. To search for a `^`, one needs to avoid writing it just after the opening bracket; to search for a hyphen `-`, one has to insert it as the last character before the closing bracket;
- besides protected special characters, there are also combinations of a backslash followed by a single letter: `\r` is a carriage-return, `\n` a newline[4], `\t` a tab char-

[3] The name "Unicode" appears nowhere in the POSIX standard, *but*, in Chapter 6, POSIX defines its meta-encoding, called *Portable Character Encoding*, based on the ISO/IEC 10646 standard, which is the ISO clone of Unicode. So it comes down to the same thing.

[4] Carriage returns were used for line separation on MacOS versions 1 to 9, newline is used for line separation in Unix systems and MacOS X. Windows uses both characters. See [10, p. 102].

acter, `\uABCD` where `ABCD` is an arbitrary 4-digit hexadecimal number, stands for the Unicode character of charcode U+ABCD (for example, `\u8A9E` stands for CJKV character "語"JA ⌊language⌋, etc.) and `\U00ABCDEF` can be used for Unicode characters outside the Basic Plane.

The second ingredient of regular expressions is parentheses and references to substrings. When a regular expression with a substring between parentheses is applied to a string, the part of the result that corresponds to the substring is stored and can be referred to by a string of the kind `\1`, `\2`, etc. Substrings are counted with respect to the left parenthesis. They can be nested. Here is an example: `([a-z])([a-z])\1\2` describes the language of words $\{c_1c_2c_1c_2\}$ with $c_1, c_2 \in \Sigma$, while the expression `(([a-z])([a-z]))\1\2\3` describes the language of words $\{c_1c_2c_1c_2c_1c_2\}$ (the first left parenthesis matches the last closing parenthesis, so that `\1` stands for c_1c_2, while `\2` stands for c_1 and `\3` stands for c_2. Regular expression engines also use the contents of substrings in the replacement part of search-replace operations.

The third ingredient of regular expressions is *quantifiers*, by which we indicate how often a given choice is allowed. In the abstract syntax, we had only two quantifiers: the unwritten one (implicitly meaning "exactly once") and the star-operator (meaning "zero, one or many times"). POSIX syntax extends those to:

- `*`: zero, one or many times;
- `+`: one or many times;
- `?`: zero or one time;
- $\{p\}$: exactly p times, where p is any natural number;
- $\{p, q\}$: minimum p and maximum q times (both are inclusive);
- $\{p,\}$: at least p times (inclusive).

(Notice, nevertheless, that p and q must be numbers explicitly given in the code, it is impossible to use a variable.)

Last, the logical OR operator can be used. It is written '`|`'. The period denotes any character besides the newline. A pair of parentheses with a question mark and a colon `(?: )` can be used whenever one wants to apply a quantifier to some part of the regular expression but without storing the result and referring to it by a `\`n notation. There are many other features of regular expressions, which we will not describe here. The interested reader can find links to documentation in the "Further Reading" section.

9.4.1.3 Lazy Quantifiers

There have been numerous extensions to POSIX syntax, such as GNU extensions, Perl extensions (PCRE), and implementations of regular expressions in all major programming languages. An impressive Web site[5] by Jan Goyvaerts compares 24 different regular expression flavors. An important example of an extension to POSIX is the case of *lazy quantifiers*, which we consider in this section.

[5] https://www.regular-expressions.info/

POSIX quantifiers are called *greedy quantifiers* because whenever they are used to detect a word belonging to the language inside a larger character string (for example, in the case of a "smart search" operation), they will always return the longest possible word. For example, if we look for a word of the language `ab+` (whose words contain an `a` followed by one or more `b`s) in the string `xxxabbbbyyy`, the regular expression engine will return `abbbb`, the string containing the maximum available number of `b` characters.

The same quantifiers can be used in *lazy* version by adding a question mark: `*?`, `+?`, `??`, `{`p`}?`, etc. These quantifiers will return the *minimal* string that belongs to the formal language of the regular expression. In the previous example of the regular expression `a+`, this time in lazy version `a+?`, applied to the string `xxxabbbyyy`, the engine will return `ab`, which contains the minimal amount of `b`s, namely one.

The reader may wonder why to bother writing `ab+?` when `ab` would return the same result. And what about the case of the regular expression `[a-z]*?` which describes… the language containing only the empty word, since the minimum of "0, 1 or many characters" is in fact 0. These examples should not fool the reader—lazy regular expressions are immensely useful when the quantified part *is surrounded by a context*. The typical example is the one of bracketing: imagine you want to replace all straight bracket pairs "[ ]" with brace pairs "{ }." The natural regular expression to do this would be `\[(.*)\]` (where the substring between parentheses will store the part between brackets to produce `{\1}` in the replacement part. If we apply `\[(.*)\]` to `[abc]de[fgh]` we will not obtain `{abc}de{fgh}` as expected but… `{abc]de[fgh}`, because the longest string matching the regular expression is returned, independently of the fact whether this string contains a closing bracket or not. The correct result is obtained by using `\[(.*?)\]`.

9.4.1.4 Regular Expressions in Python

Regular expressions in Python are very elaborate and powerful, and we refer the reader to the official documentation[6]. One needs to load the library `re`. Regular expressions can be stored in regular expression objects by the class method `re.compile` that takes a regular expression as an argument. To write regular expressions, the reader is advised to use "raw strings," which are preceded by an `r` (as in `r"(\W)ab+"`). The advantage is that no symbol needs protection and the backslash '\' becomes an ordinary character.

The regular expression package provides many Python methods. Among the most popular are

- `match`, which will apply the regular expression to the beginning of a string,
- `search`, which will all occurrences of words in the language defined by the regular expression in a string,
- `split`, which will split a string into parts according to a given delimiter, and finally,

[6] https://docs.python.org/3/library/re.html

- `sub` which will find formal words in a string and replace them with other strings.

The following example shows an example of the use of method `sub` with a function as replace argument. The purpose is to take years between 1717 and today (2023) and to add 4000 to them without changing the rest of the text:

```
1  import re,io
2  def replace(m):
3      if int(m.group(2)) >= 1717 and int(m.group(2)) <= 2023:
4          return m.group(1)+str(int(m.group(2))+4000)+m.group(3)
5      else:
6          return m.group(0)
7
8  f=io.open("input.txt",mode="r",encoding="utf-8")
9  for line in f:
10     res=re.sub(r"([^0-9])([12][0-9]{3})([^0-9])",replace,line)
11     print(res)
```

On line 2, `m` is a "match object," which results from applying a regular expression to a string. It provides `m.group(`*i*`)`, which contains the contents of the *i*-th pair of parentheses in the regular expression, and `m.group(0)` is the entire string detected by the regular expression. On line 9, the arguments of method `sub` are the regular expression, the function `replace` used for the replacement, and the string `line` to which the regular expression is applied.

9.4.1.5 Regular Expressions and ELIZA

```
(HOW DO YOU DO. I AM THE DOCTOR. PLEASE
SIT DOWN AT THE TYPEWRITER AND TELL ME
YOUR PROBLEM.) ((0 YOU 0 I 0) (PERHAPS
IN YOUR FANTASY WE 3 EACH OTHER) (DO
YOU WITH TO 3 ME) (YOU SEEM TO NEED TO
3 ME) (DO YOU 3 ANYONE ELSE))
```

Joseph Weizenbaum, *ELIZA source code*

In 1966, Joseph Weizenbaum, a professor at MIT who escaped the Nazis in 1936 as a teenager, released a program that responded roughly as would certain psychotherapists (Rogerians) [29, p. 42]. In his seminal paper, Weizenbaum justifies this choice for a program that was intended to be the first chatbot in history:

> This mode of conversation was chosen because the psychiatric interview is one of the few examples of categorized dyadic natural language communication in which one of the participating pair is free to assume the pose of knowing almost nothing of the real world. If, for example, one were to tell a psychiatrist "I went for a long boat ride" and he responded "Tell me about boats," one would not assume that he knew nothing about boats, but that he had some purpose in so directing the subsequent conversation. It is important to note that this assumption is one made by the speaker. Whether it is realistic or not is an altogether separate question. [29, p. 42]

In other words, Weizenbaum was neither a psychiatrist nor involved in psychiatry. However, he chose this conversation mode because it freed him from the necessity to include background knowledge, as we would today, using ontologies or neural models. "ELIZA" is not an acronym but refers to Eliza Doolittle, a character in Shaw's play *Pygmalion* (1913), "a barmaid in London who rose through the ranks of society in the late 19th century through marriage, various relationships, and forgeries" (Wikipedia), admirably performed by the great actress Audrey Hepburn in *My Fair Lady* (1964), an adaptation of the play, which has probably inspired Weizenbaum. His choice to name his program after an adventuress and con artist is not innocent.

We are devoting a section to ELIZA because its method of building questions out of the human's answers can elegantly be implemented using regular expressions. Indeed, there are many implementations of ELIZA, including the Python implementation by Tony Veijalainen, from which we will show some excerpts. It was developed in 2011 and based on another Python implementation by Joe Strout from 1997 (when Python was still a six-year-old toddler).

The original ELIZA code,[7] missing for over 50 years and rediscovered only in 2021 [24]. It was written in a scripting language (a scripting language is a programming language that is not compiled but interpreted, like Perl or Python) called MAD-SLIP (MAD stands for Michigan Algorithm Decoder and SLIP for Symmetric List Processor, which is, incidentally, an anagram of LISP) and invented by Weizenbaum himself [30]. The example is given in the epigraph to this section,

```
((0 YOU 0 I 0) (PERHAPS IN YOUR FANTASY WE 3 EACH OTHER) (DO YOU WISH TO
3 ME) (YOU SEEM TO NEED TO 3 ME) (DO YOU 3 ANYONE ELSE))
```

can be interpreted as follows:

- take a sentence containing some words (denoted by `0`), followed by a keyword that, after a transformation called *reflection*, becomes `YOU`, then again some words, then `I` (again after transformation), then again some words,
- reply to it by choosing randomly one of the strings that follow this expression,
- where the number indicates the token in the first expression to be used as a replacement.

For example, if the human would write "I hate you," this sentence will become, after transformation, "YOU hate I" and thus fits the pattern by taking "`() YOU (hate) I ()`." Recognizing this pattern, the script will randomly choose one of the sentences following it, for example, "`(YOU SEEM TO NEED TO 3 ME)`," which, after the replacement, becomes "`YOU SEEM TO NEED TO HATE ME`." Of course, instead of "hate," any other verb can be used (*honi soit qui mal y pense*). While the entire application is ELIZA, Weizenbaum called the script "DOCTOR."[8]

Let us see how easily this can be converted into regular expressions (Veijalainen's Python script consists of 229 lines of code, 171 of which are regular expression patterns and replacements). The main code is

---

[7] https://archive.org/details/eliza_1966_mad_slip_src/

[8] The second expression of ELIZA's greeting message is `I AM THE DOCTOR`. Does this mean that Weizenbaum was a Whovian? This is unlikely unless Weizenbaum happened to be in Canada when the series broadcasted on CBC in January 1965—the code of the greeting message was written in March 1965, as seen on `archive.org`.

```
1       print('='*72)
2       print("Hello.  How are you feeling today?")
3       s = ""
4       while s != "quit":
5           s = input(">")
6           while s and s[-1] in "!.":
7               s = s[:-1]
8           print(respond(s))
```

where `input(">")` on line 5 means that a command line prompt > is written on the console, the user writes a line of text, and this text is returned by the `input` function. Lines 6 and 7 will remove an arbitrary number of periods or exclamation marks at the end of the user's input. Then the string is given to the `respond` function.

```
1   def respond(sentence):
2       for rule, value in compiled:
3           match = rule.search(sentence)
4           if match is not None:
5               resp = random.choice(value)
6               while '%' in resp:
7                   pos = resp.find('%')
8                   num = int(resp[pos+1:pos+2])
9                   resp = resp.replace(resp[pos:pos+2], \
10                      translate(match.group(num)))
11              if resp.endswith('?.'): resp = resp[:-2] + '.'
12              if resp.endswith('??'): resp = resp[:-2] + '?'
13              return resp
```

On line 2, this function goes through the entire list of patterns. This list looks like this:

```
compiled = [(re.compile(x[0]), x[1]) for x in [
...
    ["I'm (.*)",
    [   "How does being %1 make you feel?",
        "Do you enjoy being %1?",
        "Why do you tell me you're %1?",
        "Why do you think you're %1?"]],
...
]
```

In other words, the programmer controls ELIZA's behavior by adding entries to this list. When Python reads the list, every entry is automatically regex-compiled, in the sense that the first part of the entry is compiled into a regular expression object, and the second part, which is a list of sentences with `%n` jokers, is kept unchanged. Veijalainen's code has 38 such entries, which is significantly less than in the original ELIZA, even though it is not clear out of the code how many patterns the original ELIZA had. The example we are giving is the one of the "I'm *" pattern, which, according to Baranovska & Höltgen [2, p. 9], is one of the very common sentence patterns for modern conversational assistants such as *Alexa* or *Siri*, mainly for negative conditions such as "I'm depressed." As we see in the example, there is a one-in-four chance that ELIZA will respond by "Why do you think you're depressed?"

Let us return now to the `respond` function. On line 3, each compiled regex object is used to test for the presence of a given pattern. A sentence among the ones in the second part of the entry is randomly chosen on line 5; each joker's location in the string is obtained on line 7, and the joker's value on line 8. Lines 9–10 replace a joker of value n by the n-th group extracted from the string by the regular expression. Notice that, on line 10, a function is applied to the string element corresponding to the joker, the function `translate`. Here is this function:

```
def translate(this):
    return ' '.join(reflection_of[word] if word in reflection_of \
        else word for word in this.lower().split())
```

This function uses a dictionary `reflection_of`, as follows:

```
reflection_of = {
    "am"    : "are",
    "was"   : "were",
    "i"     : "you",
    "i'd"   : "you would",
    "i've"  : "you have",
    "i'll"  : "you will",
    "my"    : "your",
    "are"   : "am",
    "you've": "I have",
    "you'll": "I will",
    "your"  : "my",
    "yours" : "mine",
    "you"   : "me",
    "me"    : "you" }
```

Whenever a word in `translate`'s input string is detected as a key of this dictionary, it is replaced by its value in the dictionary. Otherwise, the word remains unchanged.

By changing the `compiled` list and the `reflection_of` dictionary, one can build many kinds of chatbots, ELIZA is only one of them. People seemed to regard ELIZA as a genuine human interlocutor, which made Weizenbaum uncomfortable. Hofstadter (the author of *Escher, Gödel, Bach*) [11] called this the "ELIZA effect." Already in his original ELIZA paper, Weizenbaum warned against the danger of illusion:

> The whole issue of the credibility (to humans) of machine output demands investigation. Important decisions increasingly tend to be made in response to computer output. The ultimately responsible human interpreter of "What the machine says" is, not unlike the correspondent with ELIZA, constantly faced with the need to make credibility judgments. ELIZA shows, if nothing else, how easy it is to create and maintain the illusion of understanding, hence perhaps of judgment deserving of credibility. A certain danger lurks here.
>
> [29, pp. 42–43]

These words ring all the more true today in the face of ChatGPT's media frenzy.

9.4.1.6 Regular Expressions, Gender-Neutral Methods, and Poetry

Before closing the section on regular expressions, let us underline the fact that ☞ 53 graphemic gender-neutral methods are rudimentary regular expressions: in French, "étudiant·e·s"$_{\text{FR}}$ ⌊students$_{♂ \text{ or } ♀}$⌋ is `étudiant[e]?s`; in German, "StudentInnen"$_{\text{DE}}$ ⌊students$_{♂ \text{ or } ♀}$⌋ is `Student(Inn)?en`. (See also Exercise 8.4, p. 271.)

Regular expressions have also been used in experimental poetry. Here is the conclusion of a text advocating the use of regular expressions in poetry by the Kingston, PA poet, publisher, and multimedia artist Waber [28]:

> Already, I can write `/he(ck|ll)/` and you know that I mean heck and hell at the same time, and can appreciate the pun in the construction that makes heckle. Already I can write `/tim?e/` and you can apprehend in one glance the notion of being tied both by and to time. Already you can combine these two atoms of notation and take in the molecular multiple meanings of
>
> `/lip(␣s|s␣)t(i|a|u)cky?/`
>
> in a single glance (though perhaps only after you've worked it out once slowly). Already you're thinking of the problems in your own poetry, already you're wondering if they can all be solved by the right notation, already you doubt it, but already you're willing to give it a try because already you're dissatisfied with that poem you keep trying to `/w?ri(gh)?t?/`.

Besides Waber's work, there has been a performance called *Regular Expressions*[9] at the First Annual NYC Code Poetry Performance Festival, in 2014. Here is its description in the festival's program:

> *Regular Expressions* is a duet that uses a custom instrument to turn written poems into signals & sounds. A series of coded search patterns (regular expressions) are established and used to evaluate input in real-time, sending along information about the contents of the matched text. In the same way that traditional poetry takes advantage of spoken rhyme & meter, *Regular Expressions* utilizes the patterns of information hidden within the written language.

It would be hard to close this section without mentioning Ben Nadel's Annual Regular Expression Day,[10], which he has been celebrating on June 1st for the last sixteen years. Our final quote is his, and it comes from the heart:

> Regular Expressions are there for us when we need them. They don't judge us for being tired, or overworked, or distracted, or depressed. They just sit there, waiting for their moment to shine. Like a dogo or a life-long friend.

9.4.2 Finite-State Automata and Transducers

Regular expressions are human-oriented: they provide a synthetic description of a regular language most concisely. Finite-State Automata (FSA) are equivalent to regular expressions in the sense that they can describe precisely the same formal languages (namely regular languages). But FSAs are machine-oriented: they describe the behavior of a word parser, i.e., an algorithm that reads terminal symbols in linear

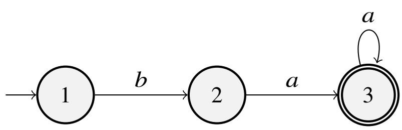
---

9 https://blog.notioncollective.com/post/80783712697/regular-expressions-nyc-code-poetry-performance

10 https://www.bennadel.com/blog/4471-the-16th-annual-regular-expression-day-june-1st-2023.htm

order and has to decide at the end of a character string whether it is, or not, a word of a given formal language.

Here is a simple example: take the language $ba+$ again. It is the language of words starting with a single b, followed by one or more a. Imagine you are a machine that has to check whether a given input string is indeed of this form. Before you receive the first letter, you will think, “Now I expect a b.” If another letter arrives, you already know the game is over. The word cannot possibly belong to the given language, so you press the big red buzzer button. But if the b arrives, you change your state of mind and switch to “now I must get an a” (the first a is mandatory; all others are optional). If no a arrives, you press the big red buzzer button. But if an a arrives, you change your expectations again and switch to “from now on, I will accept only a letters, and the word can stop at any time.” What we described as “states of mind” are the *states* of the FSA. Between states, we have *transitions*, which depend on the symbols read.



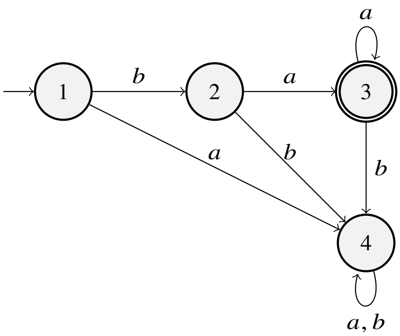
State	a	b
→1		2
2	3	
<u>3</u>	3	

Fig. 9.1 An FSA for language $ba+$ and its transition table

The reader can see a graphical representation of the FSA of the previous paragraph in Fig. 9.1. Cercles denote states and double cercles *final states*. When the word stops, we look at the current state: if it is final, the word belongs to the language. Otherwise, it does not belong to the language. This sounds much simpler than looking for derivations in a formal grammar, but then again, FSAs can only be used with regular languages, which are the simplest of all languages. The arrow on the left, coming from nowhere, denotes that this state is an *initial* one: it is the current state before we start reading symbols.

The graphical representation on the left of Fig. 9.1 is appealing. Still, its information is entirely contained in a simple table, displayed at the right of Fig. 9.1. In this table, the arrow denotes the initial state, the left column contains states, and the column under each symbol provides the states to which we transit when the symbol appears. For example, on the line of state 2, when a arrives, we move to state 3, and on the line of state 3, when another a arrives, we stay on state 3. Underlined states are final states.

This table called *transition table*, is whatever the machine needs to parse the language $ba+$. The next step is to tell the automaton what to do when a letter occurs that invalidates the string. For that, we can add an additional state (that the author calls *garbage state*) to which we re-orient transitions that invalidate the string. To be a decent garbage state, this new state (number 4 in the transition table) must not be final or have any outgoing transitions (there is no redemption: once you enter the garbage state, you stay there). The new FSA—which is now *complete*, in the sense that there exists a transition for every state and every letter symbol—and its transition table is displayed in Fig. 9.2. The transition table of a complete state has no empty cells.



State	a	b
→1	4	2
2	3	4
$\underline{3}$	3	4
4	4	4

Fig. 9.2 A complete FSA for language $ba+$ and its transition table

For completeness, let us mention that the example FSA we gave is a *deterministic FSA*. For this to happen, two conditions are necessary: there must be only one initial state, and for each symbol and each state, there can be, at most, one transition. There also exist *non-deterministic FSA*s, which can even have transitions over the empty symbol (i.e., the machine can decide to change state even without reading any symbol). Non-deterministic FSAs make it easier for humans to describe the behavior of a formal language, but they make parsing by the machine much slower. Fortunately, a *determinization algorithm* transforms a non-deterministic FSA into a deterministic one.

FSAs help describe morphological features. In Fig. 9.3, the reader can see FSAs recognizing all conjugated verb "to love" forms in the present tense and past tense in French, German, English, and Greek. The size and complexity of the FSA of each language are correlated to the complexity of the language's morphology: 22 states for French, 20 for Greek, 14 for German, and 6 for English. Notice that in German, we use empty transitions (denoted ϵ).

9.4.2.1 Finite-State Automata in Python

Implementing FSAs in Python is easy: one must write the transition table as a dictionary indexed over tuples containing state IDs and letter combinations. One can also use package *pyformlang*, which covers regular expressions, finite state automata, transducers, and formal grammars. Here is the FSA of Fig. 9.1, represented in this package:

```
import pyformlang.finite_automaton as fl

dfa = fl.DeterministicFiniteAutomaton ()
dfa.add_start_state(1)
```

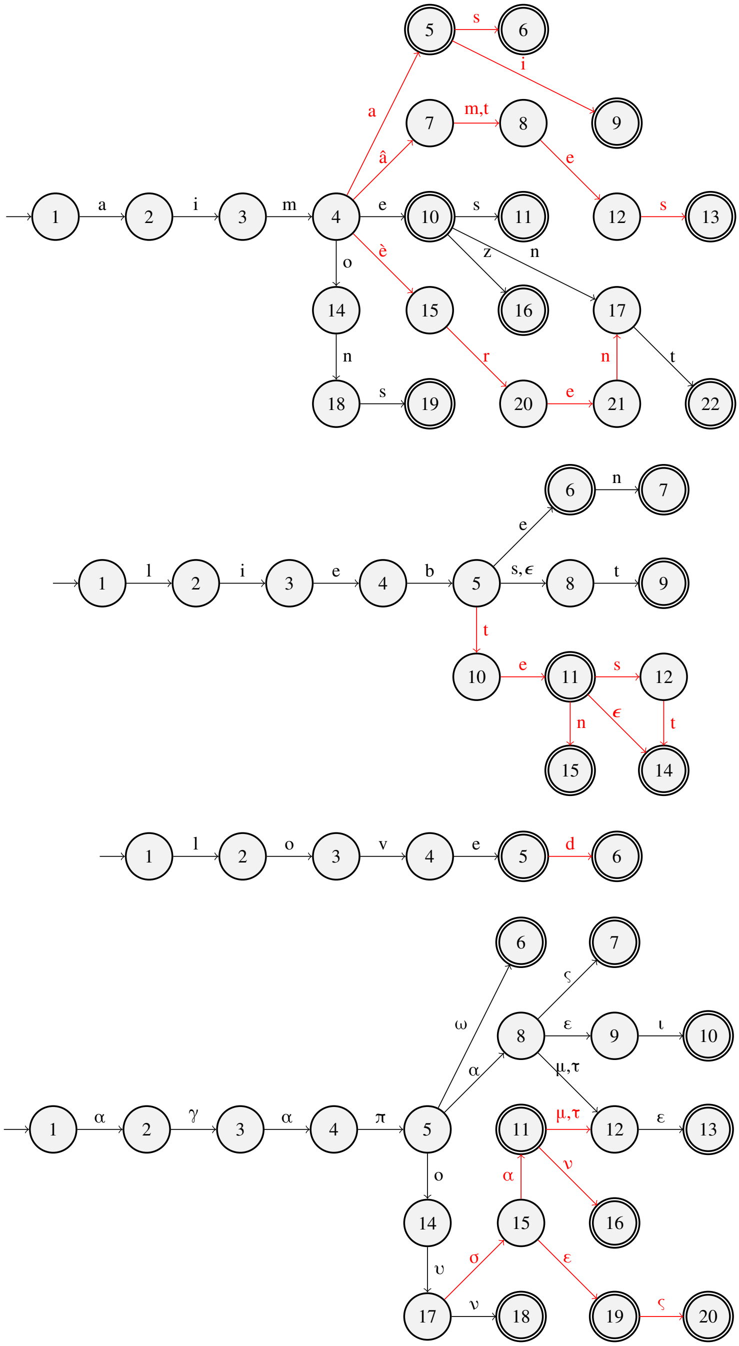


Fig. 9.3 FSAs for verbs "aimer"$_{\text{FR}}$, "lieben"$_{\text{DE}}$, "(to) love," and "ἀγαπῶ"$_{\text{EL}}$ ⌊to love⌋, in present and past tense (simple past for French, aorist for Greek). Transitions of present tense are drawn in black, and past tense transitions are in red.

```
5 dfa.add_final_state(3)
6 dfa.add_transitions([[1, "b", 2],[2, "a", 3],[3, "a", 3]])
```

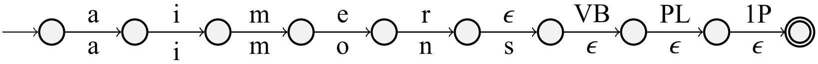
On line 4 we add an initial state, on line 5 we add a final state, and on line 6 we add transitions, which are triples consisting of departure state, symbol read, and arrival state. If we then ask whether the FSA accepts words *bba* and *baaa*,

```
print(dfa.accepts("bba"))
print(dfa.accepts("baaa"))
```

we get `False` and `True`, respectively.

9.4.2.2 Transducers

A *transducer* is an FSA with additional information: every transition on a given symbol will write a new symbol. We draw the two symbols on the two sides of the transition arrow. Transducers have been introduced by Koskenniemi [16] and used by Beesley & Karttunen [4] to represent morphology. The two sides of the arrow correspond to two types of information. To represent a given form, on the one side, one will write a lemma and morphological features (POS tag, person, number, case, ☞ 66 etc.); on the other side, one will write the letters of the form. If one of the two sides is longer, one will pad the other side with empty transitions. Here is an example, based on the 1st person of the plural or the present indicative of the French verb “aimer”$_{\text{FR}}$: the form is “aimons”$_{\text{FR}}$. On the one side, called the *lexical language*, one will place the infinitive “aimer”$_{\text{FR}}$ as well as the POS tag VB and the feature tags PL (plural) and 1P (first person). On the other side, called the *surface*, one will place “aimons”$_{\text{FR}}$. Empty transitions are added whenever necessary to produce the following transducer:



Transducers can be used in both directions: from lexical language to surface, as generators of forms, and from surface to lexical language, as morphological analyzers. In Fig. 9.4, we have represented a transducer for all forms of the present indicative of the French verb “aimer”$_{\text{FR}}$. One may wonder why to use a transducer instead of just a list of forms mapped to a list of lemmas and feature tags. The answer is that a transducer is more economical because it can be applied to unknown words. In Fig. 9.4, one can see that the letters of the stem are the same for all forms. By replacing the stem, the same transducer will work for other regular verbs, such as “poser”$_{\text{FR}}$, “traiter”$_{\text{FR}}$, “parler”$_{\text{FR}}$, “céder”$_{\text{FR}}$, etc. In the presence of an unknown verb, such as the recent “liker”$_{\text{FR}}$ ⌊to like (in social media)⌋, the transducer will be able to generate or recognize all forms since it contains the knowledge of verb conjugation.

Transducers work best for languages with mostly regular morphology. The most characteristic example that comes to mind is Arabic. Indeed, Beesley has imple- ☞ 74 mented a transducer for Arabic already in 1998 [3], and other Arabic transducers were announced in 2008 (Jaber & Delmonte [13]) and 2011 (Attia *et al.* [1]).

Besides morphology, transducers can also be used for translation [27]. In Fig. 9.5, the reader can see a transducer that translates “I am happy” into “je suis content”$_{\text{FR}}$,

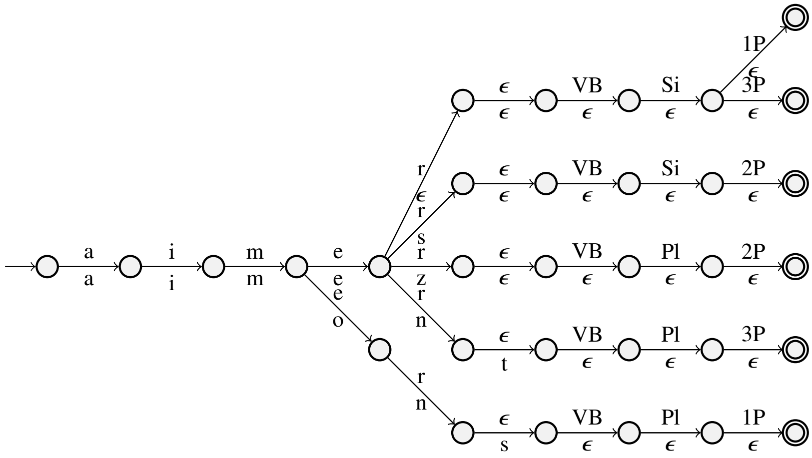


Fig. 9.4 A transducer for the present indicative tense of the French verb “aimer”$_{\text{FR}}$

and “You are happy” into either “tu es content”$_{\text{FR}}$ or “vous êtes content”$_{\text{FR}}$ or “vous êtes contents”$_{\text{FR}}$, the first being the singular (a single addressee), the second the plural of courtesy (a single addressee), and the third the plural (many addressees).

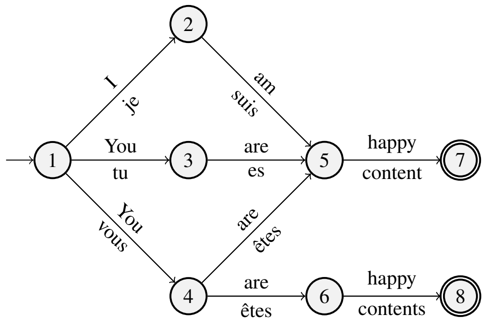


Fig. 9.5 A transducer for translating English sentences into French sentences

This transducer will give multiple correct results from English to French and a single correct result from French to English. The transducer has a double objective: by using initial and final states, it guarantees that sentences will be “words” of the formal language, that is, grammatical sentences, and by the translation feature, it allows one to switch from one language to the other.

9.4.2.3 Transducers in Python

By using the same *pyformlang* package, the transducer of Fig. 9.5 can be programmed as:

```
import pyformlang.fst as fl

fst = fl.FST ()
fst.add_start_state(1)
fst.add_final_state(7)
fst.add_final_state(8)
fst.add_transitions([[1,"I",2,["je"]],[1,"You",3,["tu"]],\
    [1,"You",4,["vous"]],[2,"am",5,["suis"]],[3,"are",5,["es"]],\
    [4,"are",6,["êtes"]],[4,"are",5,["êtes"]],\
    [5,"happy",7,["content"]],[6,"happy",8,["contents"]]])
```

This code is similar to the one of Section 9.4.2.1 except for the transitions, which are now quadruples, the second element being the lexical language item and the fourth element being a list containing surface forms. When we then ask

```
print(list(dfa.translate(["You","are","happy"])))
```

we get the result:

```
['vous', 'êtes', 'content']
['vous', 'êtes', 'contents']
['tu', 'es', 'content']
```

while the query

```
print(list(dfa.translate(["You","am","happy"])))
```

produces no result, since the sentence “You am happy” is agrammatical.

9.5 Context-Free Grammars

Context-free languages are a superset of regular languages. The prototypical irregular context-free language is

$$\{\underbrace{a\cdots a}_{n \text{ times}}\underbrace{b\cdots b}_{n \text{ times}} \text{ for } n \geq 0\}. \tag{9.1}$$

It is easy to see that an FSA cannot describe this language. If you use only one state for all the a letters and then move to a different state for the b letters, then you have no means of knowing how many a letters your word contains and, therefore, no hope of getting the same number of b letters. The only way of keeping track of the number of as is to have a state for a, another one for aa, another one for aaa, and so on, each one of these states followed by the corresponding number of bs. We need an infinite number of states to achieve this for all $n \geq 0$.

There is a method for showing that a language is not regular, called the *Pumping Lemma* [22], which states that *for every infinite regular language there are words* w_n

($n \geq 0$) of the form $w_n = x \underbrace{y \cdots y}_{n\ times} z$, *where* x, y *and* z *are fixed, and* y *is non-empty.*

It is easy to attempt to apply this lemma to language (9.1): either y contains only as, in which case the word will quickly become unbalanced, or it contains only b letters, and we have the same problem, or it contains both as and bs, in which case we have a b preceding an a, game over. An intuitive reformulation of the Pumping Lemma is that an infinite language's FSA necessarily has at least one loop.

As mentioned earlier, a context-free language can be contained in a regular language. For example, language (9.1) is contained in $L = a^*b^*$, with an arbitrary number of a and b letters.

Chomsky's constituent grammars are of the context-free type since, in the right part, they have more than one nonterminal. Here is Chomsky's context-free phrase structure grammar for the "The Doctor saved Clara from the Daleks" sentence we used in § 5.3:

S → NP VP
NP → DET NN | NN
VP → V NP PP
PP → PRE NP
DET → The | the
NN → Doctor | Clara | Daleks
V → saved
PRE → from

where the first four rules are syntactic and the four remaining ones lexical.

We have covered the various extensions to Chomsky's constituent grammars in Chapter 5. Let us see now how they have been implemented.

9.5.1 Context-Free Grammars in Python

The *nltk* package provides a module for context-free grammars that builds syntax trees for grammatical sentences. Here is the code for the grammar above, in the *nltk* package:

```
import nltk

grammar1 = nltk.CFG.fromstring("""
S -> NP VP
NP -> DET NN | NN
VP -> V NP PP
PP -> PRE NP
DET -> 'The' | 'the'
NN -> 'Doctor' | 'Clara' | 'Daleks'
V -> 'saved'
PRE -> 'from'
""")

sent = "the Doctor saved Clara from the Daleks".split()
```

```
15 rd_parser = nltk.RecursiveDescentParser(grammar1)
16 for tree in rd_parser.parse(sent):
17     print(tree)
```

and the result will be

```
(S
  (NP (DET the) (NN Doctor))
  (VP
    (V saved)
    (NP (NN Clara))
    (PP (PRE from) (NP (DET the) (NN Daleks)))))
```

The reader may notice that on line 16, the `rd_parser.parse` method returns an iterator. This is because, in the case of an ambiguous sentence, it will return more than one tree.

9.5.2 Feature-Based Context-Free Grammars in Python

Let us recall that the goal of formal grammars is not only to accept grammatical sentences, but also, and this is often way more difficult, to reject agrammatical ones. Among the necessary conditions for grammaticality, there are *valency* and *agreement*. A valency condition excludes sentences such as “I sleep an apple” since the verb “to sleep” is intransitive. As for agreement, in English, it is mainly limited to number and person, but in other languages, it also includes gender, case, and other morphological features. How can a context-free constituent avoid disagreement and valency errors? ☞ 99 ☞ 94

The first solution that comes into mind is to create special POS tags incorporating this information. For example, one could consider IV, TV, and DTV tags for intransitive/transitive/ditransitive verbs. This sounds reasonable, but for a language with three numbers, three genders, and six cases, like Russian, we would need 54 tags for all combinations, not to mention cases where multiple values are possible. This would lead to a combinatorial explosion of the number of productions needed. ☞ 66

A solution to this problem has been provided by Head-Driven Phrase Structure Grammar (§ 5.7) that uses AVMs (Attribute-Value Matrices) for attaching this kind of information. In the production rules, we can then require specific values of attributes or the equality between the attribute values of words or phrases. ☞ 107

The *nltk* package implements AVMs as features of grammar symbols. Higher-located symbols can inherit these features (going from the leaves to the root of the syntax tree) and must be *unified*. The unification of two AVMs A' and A'' is a third AVM A such that (1) every attribute included in either A' or in A'', but not in both, will be included in A, and (2) attributes included in both A' and A'' will be included in A, if and only if they have the same value in A' and A''. The various morphological features requiring agreement will be such attributes.

Let us take an example from German language: “das schöne Buch”$_{\text{DE}}$ ⌊the beautiful book⌋ (nominative), becomes “des schönen Buches”$_{\text{DE}}$ in genitive case and “das schöne Buch”$_{\text{DE}}$ in dative case. Here is a feature-based grammar that ensures agreement of case, but also gender and number:

```
import nltk

grammar1 = nltk.grammar.FeatureGrammar.fromstring("""
NP[AGR=?a] -> DET[AGR=?a] ADJ[AGR=?a] N[AGR=?a]
DET[AGR=[C=nom,G=n,N=sg]] -> 'das'
DET[AGR=[C=dat,G=n,N=sg]] -> 'des'
DET[AGR=[C=gen,G=n,N=sg]] -> 'das'
ADJ[AGR=[C=nom,G=n,N=sg]] -> 'schöne'
ADJ[AGR=[C=dat,G=n,N=sg]] -> 'schönen'
ADJ[AGR=[C=gen,G=n,N=sg]] -> 'schöne'
N[AGR=[C=nom,G=n,N=sg]] -> 'Buch'
N[AGR=[C=dat,G=n,N=sg]] -> 'Buch'
N[AGR=[C=gen,G=n,N=sg]] -> 'Buches'
""")

sent = "das schöne Buch".split()
rd_parser = nltk.parse.FeatureEarleyChartParser(grammar1)
for tree in rd_parser.parse(sent):
    print(tree)
```

and the output would be

```
(NP[AGR=[C='dat', G='n', N='sg']]
  (DET[AGR=[C='dat', G='n', N='sg']] das)
  (ADJ[AGR=[C='dat', G='n', N='sg']] schöne)
  (N[AGR=[C='dat', G='n', N='sg']] Buch))
(NP[AGR=[C='nom', G='n', N='sg']]
  (DET[AGR=[C='nom', G='n', N='sg']] das)
  (ADJ[AGR=[C='nom', G='n', N='sg']] schöne)
  (N[AGR=[C='nom', G='n', N='sg']] Buch))
```

The parser detects that both nominative and genitive cases are possible and produces the corresponding feature-enhanced syntax trees for the sentence “das schöne Buch”$_{\text{DE}}$. Furthermore, the noun-phrase NP inherits the AVM from its children. This AVM can ensure agreement with a VP. *nltk* uses the syntax `?a` to denote unification for a specific AVM.

Unification also solves the problem of underspecification: if, for example, we would like to use the word “Nichts”$_{\text{DE}}$ ⌊the Nothing⌋ (as in Michael Ende’s *The Neverending Story*), since this word is case-invariant, we would write a single rule instead of three:

```
N[AGR=[G=n,N=sg]] -> 'Nichts'
```

and unification would fill the empty slot by inferring case values from those of the article and the adjective.

9.6 Grammatical Inference

Grammatical Inference is the task of extracting the description of a formal language out of data. This is straightforward when the structure we seek needs no background knowledge, as in the case of FSA for storing the morphology of a given word. For example, by providing the data on the conjugation of verb “aimer”$_{\text{FR}}$ in French to a Python package called *dafsa* [26], as follows:

```
from dafsa import DAFSA

d = DAFSA(["aime","aimes","aime","aimons","aimez","aiment","aimai",\
    "aimas","aima","aimâmes","aimâtes","aimèrent"])

print(d)
d.write_figure("aimer-fsa.pdf")
```

we obtain the structure of a deterministic FSA and a graphical representation that the reader can see in Fig. 9.6. The stroke width of the transitions denotes the number of different word forms using them. Notice that this FSA is 32% smaller than the one we gave in Fig. 9.3, when we count the number of states.

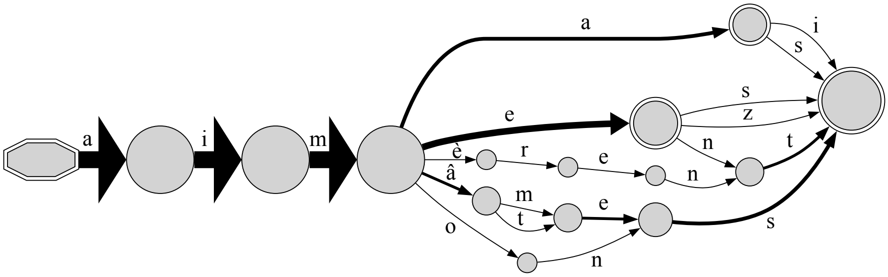


Fig. 9.6 A FSA for the present and simple past indicative tenses of verb “aimer”$_{\text{FR}}$, inferred by the Python *dafsa* package

Similarly to *dafsa*, the *scikit-sequitur* package implements the *Sequitur* [18] algorithm for extracting context-free grammars out of labeled data.

The situation is different when the grammars we seek is subject to grammaticality conditions, as when we build a grammar for a natural language. In that case, grammatical inference is performed by learning from a corpus with grammatical examples and a corpus of agrammatical examples. Children learn the syntax of their mother tongue similarly by being corrected by their parents [17, pp. 30–31]. Chomsky argued that this is a genetic component, that our brains are programmed for this task at a specific age span, and that the resources used for this task are re-oriented to other purposes when we grow up so that learning an additional language as a teenager or an adult is more difficult for us. See D’Ulizia *et al.* [6] for a survey of grammatical inference methods in Natural Language Processing.

9.7 Further Reading

9.7.1 Literature

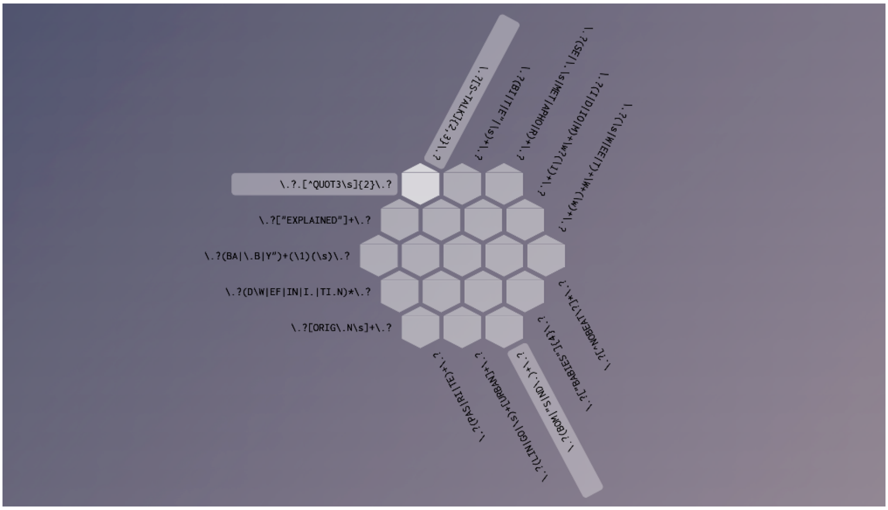


Fig. 9.7 A regular expression crossword on the *Regex Crossword* Web site

The information we gave on regular expressions is the most basic. It is sufficient for those using regular expressions in data cleaning and corpus preparation. Many more features are complex to explain and to use, but that can be very handy the day you need them (for example, there is the *negative lookahead* feature, in which you match a given string only if it is *not* followed by some other string). A very good reference for learning regular expressions is Friedl [9], translated in 7 languages.

For a fun way to practice regular expressions, we recommend the *Regex Crossword* Web site[11] by M.H. and O.B. Michelsen. On this site, you have interactive crosswords where horizontal and vertical regular expressions must match rows and columns. At the highest level, you have even hexagonal crosswords where three regular expressions match each cell. Furthermore, the authors have chosen expressions that could mislead the player, raising the challenge. The site also contains 946 (as of October 2023) puzzles submitted and ranked by community members. The reader can see the top-ranked crossword in Fig. 9.7, a real challenge. One can even buy the community's official T-shirt.

For those using regular expressions in a multilingual and multiscript environment, it may be helpful to consult the Unicode Technical Standard #18 *Unicode Regular Expressions* [7]. ☞ 343

[11] https://regexcrossword.com/

Concerning context-free grammars, Bird [5], besides being a "user's manual" for *nltk*, is also a lovely introduction to the use of context-free grammars in Natural Language Processing. (The version published by O'Reilly in 2009 is outdated. The reader can find a more recent version on the book's Web site[12].)

For the reader required to build a syntax parser from scratch without using machine learning approaches, for example, for building a natural language interface based on a controlled natural language, we recommend the *Grammatical Framework* system [23]. It is a Haskell-like functional programming language. For those unfamiliar with functional language programming, the learning curve may be steep initially. Still, once mastered, the system is quite powerful and efficient.

9.7.2 LaTeX

For drawing the automata of this chapter (except for Fig. 9.6, which was drawn by the *dafsa* Python package), we used the *tikz-automata* package[13].

This is not a widely known fact, but the latest version of LaTeX provides regular expressions for use in LaTeX documents. The syntax may be more complex than regular expressions implemented in Perl or Python, but they can be very helpful. To illustrate the use of LaTeX3's regular expressions, we will tackle a genuine LaTeX's users' need, namely *title case*. According to English typesetting traditions, words in titles must be capitalized, except for short grammatical words. Our goal will be to define a LaTeX command `\TitleCase` that will transform a sentence with capitals only where necessary (named entities, sentence begin) into a title-cased sentence. Ziuye Xiang was so kind and wrote the code for this regex-powered macro:

```
\documentclass{article}
\usepackage{expl3}

\begin{document}

\ExplSyntaxOn

\cs_new:Npn \TitleCase #1 {
  \tl_set:Nn \l_tmpa_tl {#1}
  \regex_replace_all:nnN
    {(\W|)([A-Za-z])([a-z]*)}
    {\1\c{str_uppercase:n}\cB{\2\cE}\3}
  \l_tmpa_tl
  \tl_set:Nx \l_tmpa_tl {\l_tmpa_tl}
  \regex_replace_all:nnN {(\W|)(And|In|The)(\W)}
  {\1\c{str_lowercase:n}\cB{\2\cE}\3} \l_tmpa_tl
  \tl_set:Nx \l_tmpa_tl {\l_tmpa_tl}
  \tl_use:N \l_tmpa_tl
}
```

[12] https://www.nltk.org/book/

[13] https://tikz.dev/library-automata

```
20
21  \ExplSyntaxOff
```

On lines 6 and 21, we switch the “explicit syntax” mode on and off. On line 9, the command argument is saved into variable `\l_tmpa_tl`, which will be our buffer variable (a psychologist would say: our short-term memory). The command of line 10 defines a “replace all occurrences” regex command, the first argument of which (line 11) is the regular expression in Python format (a non-letter or the empty string stored in group 1, a first letter in group 2, zero, one or more remaining letters in group 3). The second argument (line 12) is the replacement pattern. In this replacement pattern, we want to obtain `\1\str_uppercase:n{\2}\3` after expansion (`\1`, `\2` and `\3` stand for the groups we stored). To obtain this after expansion, we need to proceed with care: `\c{str_uppercase:n}` will produce the command name, `\cB{` will produce the left brace, and `\cE}` the right brace (for those familiar with TEX internals [15, p. 343], `\cB` and `\cE` assign catcodes 1 and 2 to the following character). The regular expression started on line 10 is defined and immediately applied to its third argument, which stands on line 13. Line 14 ensures that the uppercasing command we have introduced is expanded so that on line 15, all words are uppercased. Now, we have to switch grammatical words back to lowercase again. We have chosen to do it only for “and,” “in,” and “the” (the reader can add others at wish). We define a new regular expression on lines 15–16. This time, we put the grammatical word into group 2 and surrounding non-letters into groups 1 and 3 (this is to avoid lowercasing all words starting with “And,” “In,” etc.). This time, the replacement pattern contains the `\str_lowercase:n` command. Once again, the regular expression is defined and applied to `\l_tmpa_tl`, our internal “buffer” variable. Line 17 acts just like line 14: it expands the lowercasing command. Finally, line 18 will put the value of the buffer variable into the text stream, and we are done.

Writing

```
\TitleCase{I met her in the Dalek Asylum. Never saw her face,
and she died. I met her again in Victorian London, and she died.}
```

(from *The Name of the Doctor* S33E13) should produce the lovely titlecased

> I Met Her in the Dalek Asylum. Never Saw Her Face, and She Died. I Met Her Again in Victorian London, and She Died.

For more information, the reader can consult Ziyue Xiang’s Web site[14].

9.8 Exercises

Exercise 8-1: a^n, $a^n b^n$, $a^n b^n c^n$, etc.

In this exercise we will use the improper notation x^n for the word of n concatenated occurrences of x, more properly denoted by

[14] https://www.alanshawn.com/tech/2020/10/04/latex3-tutorial.html#latex3-regular-expression-xxviii

$$\underbrace{xx\cdots x}_{n \text{ times}}.$$

Find formal grammars for languages $L_1 = \{a^n \mid n \in \mathbb{N}\}$, $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$, $L_3 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$, etc.

Exercise 8-2: How many formal languages can there be?

The Oxford English Dictionary contains around 600,000 word forms. Let us take smaller languages, with at most 3,000 words. We will also limit the size of words to 10 letters and use an alphabet of 26 letters. Under these restrictions, how many formal languages can there be?

Compare this with the number of atoms we would have if, instead of the observable universe, we had a ball of iron with the same diameter.

Exercise 8-3: The complementary of a regular language

The complementary L^c of a formal language L over an alphabet Σ results from the set-theoretic difference of Σ^*, the free monoid over Σ, minus L. In other words, it is the set of all words over Σ that do not belong to L. Show that L^c is also a regular language.

Exercise 8-4: Implement French and German gender-neutral language

Write code for converting human-related nouns and adjectives into gender-neutral form in French ("écriture inclusive"$_{\text{FR}}$) and German ("binnen-I"$_{\text{DE}}$). Evaluate your code on the following set of 6+6 sentences:

Les boulangers se lèvent tôt.
Les étudiants doivent rencontrer le doyen.
Un contrôleur a contrôlé les clients.
Un jeune inspecteur passa.
Le passant hagard et l'artiste célèbre s'embrassèrent.
Les auditeurs attentifs entendirent le conférencier maladroit.

Der Professor lehrt.
Ein hiesiger Essayist und der bekannte Schriftsteller unterhielten sich.
Der Pianist verbeugte sich vor den Zuhörern.
Ich traf meinen Hausarzt auf der Straße.
Eingang verboten für Drahtseilkünstler.
Der Altphilologe war ein schöner tapferer Anarchosyndikalist.

References

[1] Mohammed Attia et al. “An Open-Source Finite State Morphological Transducer for Modern Standard Arabic”. In: *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*. 2011, pp. 125–133.

[2] “Chapter 0. TELL ME YOUR PROBLEM. Vorwort”. In: *Hello, I’m Eliza. Fünfzig Jahre Gespräche mit Computern*. Ed. by Marianna Baranovska and Stefan Höltgen. projekt verlag, 2005, pp. 9–20.

[3] Kenneth R. Beesley. “Arabic Morphology Using Only Finite-State Operations”. In: *Computational Approaches to Semitic Languages*. ACL, 1998, pp. 50–57.

[4] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, 2003.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O’Reilly, 2009.

[6] Arianna D’Ulizia, Fernando Ferri, and Patrizia Grifoni. “A survey of grammatical inference methods for natural language learning”. In: *Artificial Intelligence Reviews* 36 (2011), pp. 1–27.

[7] Mark Davis. *Unicode Regular Expressions*. Vol. 18. Unicode Technical Standards. https://www.unicode.org/reports/tr18/. 1998–2022.

[8] Patrick Dehornoy. *Mathématiques de l’informatique*. Paris: Dunod, 2000.

[9] Jeffrey E.F. Friedl. *Mastering Regular Expressions*. 3rd ed. O’Reilly, 2006.

[10] Yannis Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O’Reilly, 2007.

[11] Douglas R. Hofstadter. “Preface 4. The Ineradicable Eliza Effect and Its Danger”. In: *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, 1996, p. 155.

[12] “IEEE Std 1003.1-2017. The Open Group Base Specifications Issue 7”. https://pubs.opengroup.org/onlinepubs/9699919799. 2018.

[13] Suhel Jaber and Rodolfo Delmonte. “Arabic Morphology Parsing Revisited”. In: *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander Gelbukh. Springer, 2008, pp. 96–105.

[14] Stephen C. Kleene. “Representation of Events in Nerve Nets and Finite Automata”. RAND Corporation Report R4-704 https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf. 1951.

[15] Donald E. Knuth. *The TEXbook*. Vol. A. Computers & Typesetting. Addison-Wesley, 1986.

[16] Kimmo Koskenniemi. *Two-level morphology: A general computational model for word-form recognition and production*. Publication 11. University of Helsinki, 1983.

[17] Barbara Lust. *Child Language: Acquisition and Growth*. Cambridge University Press, 2007.

[18] Craig G. Nevill-Manning and Ian H. Witten. “Identifying Hierarchical Structure in Sequences: A linear-time algorithm”. In: *Journal of Artificial Intelligence Research* 7 (1997), pp. 67–82.

[19] Catarina Dutilh Novaes. *Formal Languages in Logic. A Philosophical and Cognitive Analysis*. Cambridge University Press, 2012.

[20] Barbara H. Partee, Alice ter Meulen, and Robert E. Wall. *Mathematical Methods in Linguistics*. Dordrecht: Kluwer, 1990.

[21] Geoffrey K. Pullum. *The great Eskimo vocabulary hoax, and other irreverent essays on the study of language*. Chicago, IL: University of Chicago, 1991.

[22] Michael Rabin and Dana Scott. “Finite Automata and Their Decision Problems”. In: *IBM Journal of Research and Development* 3 (1959), pp. 114–125.

[23] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. Stanford: CSLI Publications, 2011.

[24] Jeff Shrager. “Joseph Weizenbaum’s Original ELIZA”. https://sites.google.com/view/elizagen-org/the-original-eliza. 2021.

[25] Dan A. Simovici and Richard L. Tenney. *Theory of Formal Languages with Applications*. World Scientific, 1999.

[26] Tiago Tresoldi. In: *The Journal of Open Source Software* 5 (2020). https://doi.org/10.21105/joss.01986.

[27] Stephan Vogel and Hermann Ney. “Translation with Cascaded Finite State Transducers”. In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. USA: Association for Computational Linguistics, 2000, pp. 23–30.

[28] Dan Waber. “Regular Expressions as a System of Poetic Notation”. In: *P-Queue* 5 (2008), pp. 143–156.

[29] Joseph Weizenbaum. “ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine”. In: *Communications of the ACM* 9 (1966), pp. 36–45.

[30] Joseph Weizenbaum. “Symmetric Lisp Processor”. In: *Communications of the ACM* 6 (1963), pp. 524–536.

Chapter 10
Logic

SPOCK: In this situation, logic dictates…
MCCOY: Logic? Oh, my God! There's a maniac trying to make us blow up our own damn ship, and you talk about logic?

Star Trek, *Into Darkness*

According to Aristotle, to be a good speaker, one has to excel in the three principles of rhetoric: *logos* (correct reasoning), *ethos* (good character and credible arguments), and *pathos* (emotion). This chapter will focus on the first principle, *logos*, and the discipline that studies it, namely *logic*.

Logic has evolved a lot in the twentieth century, and Artificial Intelligence has used it from the beginning to represent information and knowledge. We will call the main kind of logic, the one that resulted from contributions by Frege, Peirce, Tarski, Gödel, Carnap, and Quine, *First-Order Logic* (FOL). It is also known as *Predicate Calculus* or as *Mathematical Logic*. After a thorough description of FOL's workings, we will see that other logics have been defined in Artificial Intelligence, either to simplify the task of automatic reasoners (i.e., programs that reason) or add additional features to FOL. We speak of *logic fragments* in the former case. A particularly important case for Natural Language Processing is the one of *Description Logics*. In the latter case, we speak of *Higher-Order Logics* and of *Modal Logics*.

All logics share the following properties:

1. They are *formal languages*, the "words" of which we call *statements* (or *formulas*). Let us denote a logic by L and an arbitrary statement by F ($F \in L$). ☞ 245
2. Some statements of each logic (usually a very small number among them) are called *axioms*. Let us denote by A_i the axioms of L.
3. Every logic L has a set $\{R_1, \ldots, R_n\}$ of applications $R_i\colon L^{p_i} \to L$ ($p_i \geq 1$), called *inference rules*. Each inference rule R_i may have a different p_i, called the number of *premises*. The statement resulting from the inference rule is called its *conclusion*.

To give an example, which we will formalize in a moment, in FOL, we can write statements with meanings such as "Socrates is a man" and "All men are mortals."

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_10

An inference rule called *modus ponens* will take these two statements as premises and produce a third statement, whose conclusion is "Socrates is mortal." The action of applying an *inference rule* is called an *inference* (for millennia, it was called a *syllogism*). As for our example of Socrates's mortality, it is the kind of example found in all textbooks worldwide. According to Wheeler [35], it first appeared in a book by John Stuart Mill, in 1843 [32, p. 228].

The notation used for inference is

$$F_1, F_2, \ldots, F_n \underset{r}{\vdash} F',$$

i.e., "from formulas $F_1, F_2, \ldots, F_n$ we infer formula F' through inference rule r." The notation

$$\frac{F_1 \qquad F_2 \qquad \cdots \qquad F_n}{F'}\; r$$

is also used.

By applying inferences to axioms, we get new statements. We get even more new statements by applying inferences to these statements and/or axioms. The *inferential closure* of a set of axioms and a set of inference rules (i.e., the set we obtain by endlessly applying inference rules to axioms and resulting statements) is called a *theory*. Any statement of a theory is either an axiom or a *theorem*. One *proves* a theorem by showing how it can be obtained from axioms (or, more probably, from other theorems that have been proved). The reader may argue, "I spent years proving theorems in school, but I never heard of axioms of mathematics." In fact, they do exist. They are part of a discipline called *Axiomatic Set Theory* (see Takeuti & Zaring [33][1]), which is part of the curriculum of Pure Mathematics at the upper-undergraduate level.

First-Order Logic for Natural Language Processing will heavily use inference, but not at the axiomatic level. Inference corresponds to a specific *mode of reasoning*, called *deduction*, based on *modus ponens*: if A is true and A implies B, then B is true. There are two other modes of reasoning: first, *abduction*, used by Sherlock Holmes [5]: when B is true, and A implies B, then A is probably true as well, or at least deserves to be considered as a hypothesis. Finally, in *inductive* reasoning, one starts with observations, builds patterns, hypotheses, and finally, a theory, which has to be proved. Data science is abductive.

Let us now take a closer look at First-Order Logic.

[1] Not only does Takeuti & Zaring [33] deal with the very foundations of mathematics, since it explores Axiomatic Set Theory, but it is also the first volume of the prestigious *Graduate Texts in Mathematics* series that has accompanied and still accompanies the formative years of hundreds of thousands of mathematicians.

10.1 First-Order Logic

10.1.1 Formal Theory

The fundamental element of First-Order Logic (FOL) is the *statement* (or *formula*). As our goal is to describe the world by representing information and knowledge, besides using semantically irrelevant symbols such as $f, g, \ldots$ or $P, Q, \ldots$, we will also use symbols referring to objects or situations of the world we want to describe. For example, to model the situation that River is Amy's daughter and that Amy loves Rory, we will write the formulas $\mathsf{River} = \mathsf{daughter}(\mathsf{Amy})$ and $\mathsf{loves}(\mathsf{Amy}, \mathsf{Rory})$. The symbols daughter and loves are different: the former returns a human being, which happens to be River, while the latter asserts a fact. We will call the former a *function* and the latter a *predicate*. As for River, Amy, and Rory, they refer to individuals of the world we want to describe. We will call them *constants*.

What we also need is a way to combine formulas, e.g., to say that Alice is a woman *and* Alice plays chess. This operation is called a *conjunction*, and the symbol we will use is $\wedge$. There is also an "or" operation, called *disjunction*, denoted by the symbol $\vee$. These symbols are called *connectives*. And to say that Alice does not play chess, we need a symbol for *negation*, it is $\neg$.

Using negation and disjunction, we build a new connective called *implication*, denoted $\rightarrow$. The definition follows: $A \rightarrow B$ is the same as $\neg A \vee B$. Double implications (i.e., implications in both directions) are denoted by $\leftrightarrow$.

To assert general rules (such as "all men are mortal"), we need *variables* and *quantifiers*. Variables are denoted by letters $X, Y, \ldots$, and quantifiers are written as in mathematics: $\forall$ is the *universal quantifier*, and $\exists$ is the *existential quantifier*. Note that there is no "$\exists!$," to express uniqueness. Instead, we will use the equality relation as follows: to say that there exists a unique X such that $P(X)$, we will write

$$\exists X\, P(X) \wedge (\forall Y\, P(Y) \rightarrow X = Y),$$

in other words, there exists a value of X such that

1. we have $P(X)$; let us call this value of X, X_0,
2. and such that if for any value of Y we have $P(Y)$, then this value of Y is equal to X_0.

To summarize, here is the complete list of ingredients of FOL:

- *predicates* $P, Q, \ldots$ of a given *arity* (the number of arguments);
- *functions* $f, g, \ldots$ of a given arity;
- functions of arity 0 are called *constants*;
- *variables* $X, Y, \ldots$;
- *connectives* $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$;
- *quantifiers* $\forall$, $\exists$;
- the sign of *equality* $=$;
- parentheses.

Functions and predicates take a fixed number of arguments. This number is called *arity* of the function or predicate. One can say that a function of arity 0 is a constant.

To these elementary building blocks, we add another building block obtained recursively:

- a *term* is either a constant, a variable, or a function applied to one or more terms. For example, if X, Y, Z are variables and f, g, h are functions, then $f(g(X), h(Y), Z)$ is a term.

The arguments of predicates are terms. For example, $f(g(X), h(Y), Z)$ can be the argument of P, so that $P(f(X), h(X), Z)$ is a predicate.

Formulas are also defined recursively: a predicate is a formula, and any combination of formulas using connectives is a formula. E.g., $P(X, g(Y))$ is a formula, and so is $P(X) \wedge (\neg Q(Y, Z))$.

Quantifiers are always applied to variables. They have a *scope*: a variable is quantified by application of the quantifier, and its scope is whatever follows. For example, in $P(X) \wedge \forall Y\, Q(Y)$, the scope of $\forall Y$ is $Q(Y)$. So if we write $P(Y) \wedge \forall Y\, Q(Y)$, then $P(Y)$ is outside the scope of the quantifier, which means that we have called "Y" two different variables, one that is not quantified (we call it a *free* variable) and one that is quantified (we call it a *bound* variable). It is authorized but considered a bad practice to give the same name to two different variables. The fault can easily be corrected by changing the symbol used for the bound variable, e.g., in our example, we get $P(Y) \wedge \forall X\, Q(X)$.

A formula with no free variable is called a *closed formula*.

Let us attempt now to formalize the "All men are mortals" syllogism. The first formula (also called *major premise*) is $\forall X, \mathsf{man}(X) \longrightarrow \mathsf{mortal}(X)$. The second formula (*minor premise*) is $\mathsf{man}(\mathsf{Socrates})$. Obviously, man and mortal are predicates, X a variable, and $\mathsf{Socrates}$ a constant. We now wish to apply modus ponens, namely

$$\frac{A \to B \qquad A}{B}\ \text{modus ponens}$$

But there is a hitch. Our first formula deals with the quantified variable X, while the second one deals with the constant $\mathsf{Socrates}$. We need yet another transformation that will replace the variable X by the constant $\mathsf{Socrates}$.

For that, we will introduce a transformation of formulas called *substitution*. If F is a formula, X a variable, and T a term, then $\mathsf{Subst}(\{X/T\}, F)$ will be a new formula, in which T has substituted all occurrences of X.

Now we have the tool to transform the X of the first formula into the $\mathsf{Socrates}$ constant of the second, but who gives us the right to do so? The answer is that besides modus ponens, there is another inference rule we have the right to use, the *rule of universal instantiation*:

$$\frac{\forall X\, F}{\text{Subst}(\{X/T\}, F)}\ \text{universal instantiation}$$

where F is a formula, X a variable, T a term. The term "universal instantiation" may sound like the ultimate truth of some religious sect. Still, the rule is quite simple: it boils down to "if we can assert something for *all* values of some variable, then we can assert it for *some* values of the variable."

In our case, we will replace X by $\mathsf{Socrates}$, and the trick is done by applying the following chain of inferences:

$$\dfrac{\dfrac{\forall X\,\mathsf{man}(X) \rightarrow \mathsf{mortal}(X)}{\mathsf{man}(\mathsf{Socrates}) \rightarrow \mathsf{mortal}(\mathsf{Socrates})}\text{ univ. inst.}\quad \mathsf{man}(\mathsf{Socrates})}{\mathsf{mortal}(\mathsf{Socrates})}\text{ modus ponens}$$

10.1.2 Model Theory

In the previous section, we gave ourselves the means to write formulas. But what is the link between these formulas and the world we want to describe? This is where *model theory* steps in.

Let us take, for example, the world we live in and, more specifically, all living creatures. There is, apparently, only one biologically immortal creature: the jellyfish *Turritopsis dohrnii* [26], let us call it "t.dohrnii" for short. This creature is immortal because it can become younger at any time and wish. But no man is a t.dohrnii (by "man," we mean man or woman). Therefore, we can reasonably assert that all men are mortal (including Elvis) and that those living creatures that are not men (or women) are not necessarily mortal (for example, t.dhornii is not mortal). We can say that every living creature is either a man or is not a man. As all men are mortal, we can say that every living creature is either mortal or not a man (because if it's not a man, then we can't say for sure whether it is mortal; after all, it may be a t.dhornii). In other words, the formula $\forall X\, \neg\mathsf{man} \vee \mathsf{mortal}$ is *true* in the world we live in. And, by definition of the $\rightarrow$ connective, this formula is identical to the formula $\forall X\,\mathsf{man}(X) \rightarrow \mathsf{mortal}(X)$.

What has changed vs. the previous section? There, we were writing formulas and inferring new formulas from previous formulas. Here, we choose a *domain* for the world we are interested in (which may be the real world, or a world where Gallifrey exists, or Athens of the 5th c. BCE so that Socrates is still alive, etc.) and we assert that in this domain a given formula is (or is not) *true*. This is what we did in the previous paragraph: from our knowledge of the biology of the species, we concluded that $\forall X\,\mathsf{man}(X) \rightarrow \mathsf{mortal}(X)$. Most of the time, to assert the truth of a formula in some domain, we need knowledge about the domain—the truth of the formula is, in fact, part of the knowledge we have of the domain.

Let us be more precise about applying a formula to a domain. For this, we have two operations:

1. *interpretation* (also called *valuation* by some authors), which we denote by $\mathbf{I}$ (used as a superscript $^{\mathbf{I}}$), and
2. *assignment*, denoted by σ, and needed only when the formula has free variables.

In logic (not only FOL but any logic), the operations of "interpretation" and "assignment" have very specific technical meanings.

Interpretation is a process consisting of the following steps:

1. a *domain* D is chosen;
2. for every constant C of the formula, a value for $C^{\mathbf{I}}$ is chosen in D;
3. for every variable X of the formula, $X^{\mathbf{I}}$ is a variable taking values in D;
4. for every n-ary predicate P of the formula, $P^{\mathbf{I}}$ is a function sending D^n into $\{\top, \bot\}$ where $\top$ denotes "true" and $\bot$ denotes "false" (these symbols are not part of the domain, they are hard-coded in model theory);
5. for every n-ary function f of the formula, $f^{\mathbf{I}}$ is a function from D^n into D;
6. to calculate the interpretation of a term, we use the fact that the interpretation of a composition of functions is the composition of the interpretation of the functions, i.e., $(f(g(X)))^{\mathbf{I}}$ is $f^{\mathbf{I}} \circ g^{\mathbf{I}}(X^{\mathbf{I}})$;
7. for every equality relation between terms $T = T'$, $(T = T')^{\mathbf{I}}$ is true iff $T^{\mathbf{I}}$ and $T'^{\mathbf{I}}$ denote the same element of D;
8. finally, in case of a connective between two formulas, interpretation is calculated as follows:
    - $(\forall X\, P(X))^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}(X^{\mathbf{I}})$ is true for all values of of D taken by $X^{\mathbf{I}}$;
    - $(\exists X\, P(X))^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}(X^{\mathbf{I}})$ is true for at least one value of D taken by $X^{\mathbf{I}}$;
    - $(\neg P)^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}$ is false,
    - $(P \wedge Q)^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}$ and $Q^{\mathbf{I}}$ are both true,
    - $(P \vee Q)^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}$ is true or $Q^{\mathbf{I}}$ is true (or both),
    - and therefore, $(P \rightarrow Q)^{\mathbf{I}}$ is true iff $P^{\mathbf{I}}$ is false or $Q^{\mathbf{I}}$ is true,
    - and therefore, $(P \leftrightarrow Q)^{\mathbf{I}}$ is true iff either $P^{\mathbf{I}}$ and $Q^{\mathbf{I}}$ are both true or if they are both false.

Let us describe our interpretation of the formula $\forall X\, \mathsf{man}(X) \rightarrow \mathsf{mortal}(X)$.

- Step 1: we choose a domain: let D be the real world we live in, with our knowledge of it.
- Step 2 is not applicable since this formula has no constants.
- Step 3: The variable $X^{\mathbf{I}}$ will take values in D.
- Step 4: let us choose functions $D \rightarrow \{\top, \bot\}$ for the interpretations of our two predicates: $\mathsf{man}^{\mathbf{I}}$ and $\mathsf{mortal}^{\mathbf{I}}$. The first one will be the function that is true for all men (and women), and the second will be the function that is true for all mortal entities in the real world (in particular, this function will send t.dohrnii to false).
- Steps 5, 6, and 7 are not applicable.
- Finally, according to step 8, $(\forall X\, \mathsf{man}(X \rightarrow \mathsf{mortal}(X)))^{\mathbf{I}}$ will be true iff for each $X^{\mathbf{I}}$ in D, either $\mathsf{man}^{\mathbf{I}}(X^{\mathbf{I}})$ is false or $\mathsf{mortal}^{\mathbf{I}}(X^{\mathbf{I}})$ is true.

This means that our formula will be true in our interpretation if every entity of our world either is not a man or is mortal. And this is indeed the case, according to our knowledge of the world.

When the interpretation of a formula is true, this interpretation is called a *model* of the formula. Our formula $(\forall X\, \mathsf{man}(X) \rightarrow \mathsf{mortal}(X))^{\mathbf{I}}$ would be false in domain D', where D' is the fictional world of the movie *Highlander* (1986) since there would be at least one element of D that would be both man and not mortal (namely Connor

MacLeod). In other words, a given formula can be true or false, depending on our chosen interpretation.

In some cases, we can assert that a formula is true *independently of the interpretation*. This is, for example, the case of $\forall X\, P(X) \vee \neg P(X)$. We call such a formula a *tautology*. And a formula like $\forall X\, P(X) \wedge \neg P(X)$, which is always false, is called a *contradiction*. A formula with at least one model is called *satisfiable*.

Let us return to our "all men are mortal" syllogism. Imagine we choose an interpretation in which the premises are true: all men are mortal, and Socrates is a man. Will we necessarily have that Socrates will be mortal? Can we be sure about this? After all, our interpretations can be weird at will…

The good news is that according to Gödel's completeness theorem [19, §5.2.4], this is indeed the case for FOL. If, in some interpretation, the premises of an inference are true, then the conclusion will be true as well in that interpretation. In other words, every model of the premises is also a model of the conclusion. When this happens, we say that the conclusion is a *consequence* of the premises, and Gödel's theorem says precisely that, in FOL, consequence is equivalent to deduction.

We will use the following notation: if $\mathbf{I}$ is a model of a formula F, we will write $\mathbf{I} \models F$. And if F' is a consequence of F, then we will write $F \models F'$. Gödel's completeness theorem says that $\models$ equals $\vdash$.

As for *assignment*, it assigns a value to each free variable. This is necessary to obtain a truth value for a given formula in a given interpretation. Otherwise, the truth value of the formula will depend on its variables (unless, of course, the formula is a tautology or a contradiction).

10.1.3 Propositional Logic

> the rules of chess occupy 10^0 pages in first-order logic, 10^5 pages in propositional logic, and 10^{38} pages in the language of finite automata.
>
> Stuart Russell, *Unifying Logic and Probability*

Propositional Logic is a fragment of First-Order Logic, based on propositions and *connectives*. If we remove variables, functions, constants (in other words, terms) from First-Order Logic formulas, and necessarily also quantifiers (since they are applied to variables) and equality (which is applied to terms), what remains? Predicates can have no arguments anymore (since they would be terms), so we can keep only predicates of arity 0. A predicate of arity 0 is called a *proposition*. When interpreted, a proposition is sent to a value "true" or "false." One can choose any domain, imaginative at wish, but the interpretation ultimately boils down to binary truth values for propositions. We can build tables where propositions take truth values. Every such combination is an interpretation. If the global formula is true, then the combination of truth values of propositions is a model. Such a table is called a *truth table*. Here is an example: if we take the formula $P \to (Q \vee R)$, its truth table will be

P	Q	R	$P \to (Q \vee R)$	P	Q	R	$P \to (Q \vee R)$
$\top$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$

This table represents *all* possible interpretations of the formula. There are eight of them (that is, 2^n for $n = 3$) propositions, and all but one are models. Except for one line, the formula could have been a tautology.

10.1.4 Natural Language Formalization

It is well known that FOL cannot represent all natural language sentences; we will see some examples. So, it makes little sense to take a natural language corpus and try to convert it into FOL, especially if the texts are informal, such as a corpus of tweets.

But it makes sense to convert natural language into logic if we know in advance that the text will be well-written and unambiguous, for example, in the case of a
☞ 197 controlled natural language or a question-answer system with stringent rules on the language used on the human side.

In this section, we will consider some generic examples.

Constants will usually represent the world's abstract or concrete entities. Verbs used to assert a state of the world will usually be represented by predicates, and their arguments will become arguments of the predicate. For example, to state that "the Doctor travels using a Tardis," the formula would be $travels(\mathsf{Doctor}, \mathsf{Tardis})$, assuming that the first argument is the subject of the verb and the second is the means of transportation. Of course, these are not the only information elements that come into play when discussing travel. There is also the origin, the destination, the date, the duration, and so on. We will see in Chapter 11 how to deal with this problem more efficiently. For the moment, we will consider that we will redefine the semantics of each predicate depending on our needs in a specific context.

Negation in natural language is represented by logical negation in FOL.

Connectives $\wedge$ and $\vee$ are used to represent the natural language connectives "and" and "or," even though the natural language connectives often carry much more information than their logical counterparts, for example, in the sentence "I ate and
☞ 181 went to bed," the use of the connective implies a temporal order.

The connective $\to$ is used for sentences of the "if... then..." kind. Typically, a sentence like "If the ball is green, then it is not black" will be translated by $\mathsf{green}(\mathsf{ball}) \to \neg\mathsf{black}(\mathsf{ball})$.

The example we just used is interesting because saying "the ball" implies that we have some means of identifying the ball we are talking about, either by *deixis* (pointing at it with our finger while uttering the sentence) or by implicit reference to a referent in a previous sentence. As this sentence is true in general and not only for a specific ball, it would be more natural to say, "Whenever a ball is green, it is not black," or "Every ball that is green is not black."

Quantified sentences using the determiner "every" are represented by logical formulas using a universal quantifier (and some dummy variable) and an implication. For example, "every green ball is not black" is written

$$\forall X\, \mathsf{ball}(X) \wedge \mathsf{green}(X) \rightarrow \neg \mathsf{black}(X).$$

The reader may wonder whether it is necessary to specify that X must be a ball explicitly. This is where interpretation comes into play. When we formalize a natural language sentence into a formula to describe the world, the intended interpretation counts as much as the formula itself. If we choose the domain D of balls, then the predicate $\mathsf{ball}(X)$ is a tautology since it is true for every value of X. But if we choose, more generally, the entire world we live in as domain, then we have to explicitly request that X is a ball because there are other entities in the world than balls.

The reader should note that the choice of the implication connective is very important. Replacing the implication by, for example, a conjunction has an entirely different outcome: the formula

$$\forall X\, \mathsf{ball}(X) \wedge \mathsf{green}(X) \wedge \neg \mathsf{black}(X)$$

can be translated in natural language as "all objects in my world are balls that are green and that are not black," which is not what we wanted to say (and is false in the domain of the real world).

Sentences starting with "There is...," or "There exists...," are formalized using the existential quantifier (over some dummy variable) and the conjunction connective. For example, to say that "there is an immortal animal," we will write

$$\exists X\, \mathsf{animal}(X) \wedge \neg \mathsf{mortal}(X).$$

Once again, the choice of connective is very important. Using an implication instead of a conjunction, i.e., writing $\exists X\, \mathsf{animal}(X) \rightarrow \neg \mathsf{mortal}(X)$, would mean "there is an X which either is not an animal or is an immortal animal," which is less interesting since there are zillions of entities in the real world that are not animals (chairs, ideas, the fifth floor of the Empire State Building, David Beckham's left foot, etc.).

One should also consider the fact that existence does not imply uniqueness. This is why some authors recommend translating "$\exists X\, P(X)$" by "for *some* X we have $P(X)$," so that the exact cardinal of these values remains unspecified.

Also, one should not assume that $\forall X\, P(X)$ necessarily implies $\exists X\, P(X)$. If this were true, then every model of the former would also be a model of the latter formula. Take an interpretation with an empty domain. Then, by convention,[2] $\forall X\, P(X)$ is

---

[2] In an empty domain, there is no value to which we can apply the predicate and find out whether it is true or false. So, how can we assert whether the predicate is true or false? This sounds like a *koan*, which is a paradoxical story, the purpose of which is to lead us to sudden enlightenment (*satori*) in Zen Buddhism. Jean-Louis Lebris de (Jack) Kérouac experienced the satori in 1965, as he accounts in *Satori in Paris*, thanks to a French taxi driver named Raymond Baillet, or maybe by his paranoiac fear in the foggy streets of Brest, Brittany, at 3 AM. Unrelated to Kérouac and

true. At the same time, $\exists P(X)$ requires the existence of at least one element in the domain and therefore is false.

When multiple quantifiers are necessary, one must be careful to use them in the correct order. Often, this amounts to disambiguating a sentence. A typical example is the Blues Brothers' song "Everybody Loves Somebody." Is this somebody the same for everyone, or does everyone love somebody else? The former would be represented as

$$\exists Y\, \forall X\, \mathsf{loves}(X, Y)$$

and the latter would be represented by

$$\forall X\, \exists Y\, \mathsf{loves}(X, Y).$$

This comes from the fact that whenever we enter the scope of a quantifier for a given variable X, the values of a subsequently quantified variable Y will depend on the value of X.

When formalizing natural language sentences, we often have to choose between functions or predicates. For example, the sentence "Clara is the Doctor's companion" can be translated into

$$\mathsf{Companion}(\mathsf{Doctor}, \mathsf{Clara})$$

or

$$\mathsf{Companion}(\mathsf{Doctor}) = \mathsf{Clara}.$$

In the first case, $\mathsf{Companion}$ is a predicate, in the second, a function. These formulas are perfectly equivalent (in that they have precisely the same models), as can be shown by simple set theory on the interpretation level.

In the following section, we will give an example of natural language sentence formalization and reasoning. It is not just any example but a (pseudo-)paradox that has caused much ink to flow.

10.1.4.1 An Example: The Barber's (Pseudo-)Paradox

A famous logical paradox is the so-called Barber's Paradox: *A village's barber shaves only those men of the village that do not shave themselves, must he shave himself or not?*

According to Raclavský [28]:

> One of the paradoxical aspects of the Barber paradox is that it is not a paradox, though many people still think it is.

It ceases to be a paradox when one remarks that such a barber cannot exist. Let us prove this by FOL. Saying that there is a barber that shaves all men of a given village,

---

unaffected by the fog in Brest, the philosopher Quine gives various arguments in favor of the truth of universal quantifiers in empty domains, which he calls *vacuous quantification*, in [27].

assuming that our domain is the set of the men of this village (so that we need no predicate to restrain ourselves to this set), would be

$$\forall y\ \mathsf{Shaves}(\mathsf{Barber}, y).$$

Adding now the condition that the barber shaves those who do not shave themselves, we get

$$\forall y\ (\neg\mathsf{Shaves}(y, y) \longrightarrow \mathsf{Shaves}(\mathsf{Barber}, y)).$$

But this allows the barber to shave other men who potentially do not shave themselves. To specify that the barber shaves *only* the men who do not shave themselves, we need a second condition:

$$\forall y\ \big((\neg\mathsf{Shaves}(y, y) \longrightarrow \mathsf{Shaves}(\mathsf{Barber}, y) \wedge \mathsf{Shaves}(y, y) \longrightarrow \neg\mathsf{Shaves}(\mathsf{Barber}, y)\big).$$

It is easy to show that this formula is always false. Indeed, by replacing implications with disjunctions, our formula becomes:

$$\forall y\ \big((\mathsf{Shaves}(y, y) \vee \mathsf{Shaves}(\mathsf{Barber}, y)) \wedge (\neg\mathsf{Shaves}(y, y) \vee \neg\mathsf{Shaves}(\mathsf{Barber}, y))\big).$$

The formula contains a universal quantifier applied to y; therefore, to show it is false, we need only to find a specific value of y for which it is false. Let us take $y = \mathsf{Barber}$. We then get

$$\big(\mathsf{Shaves}(\mathsf{Barber}, \mathsf{Barber}) \vee \mathsf{Shaves}(\mathsf{Barber}, \mathsf{Barber})\big)$$
$$\wedge \big(\neg\mathsf{Shaves}(\mathsf{Barber}, \mathsf{Barber}) \vee \neg\mathsf{Shaves}(\mathsf{Barber}, \mathsf{Barber})\big).$$

In other words, it is of the form $(\alpha \vee \alpha) \wedge (\neg\alpha \vee \neg\alpha)$, that is, $\alpha \wedge \neg\alpha$, and thus a contradiction. Since it is false for one value, it is not true for all values, and hence, the universally quantified formula is false.

Let us use a Python program to prove that there can be no barber as described in the Barber's Paradox. We will use a package called *z3* (beware, it must be installed by

```
pip install z3-solver
```

even though in the code we import `z3`). This package is an interface to a reasoner called *Z3*,[3] which must be installed before installing the Python package.

Z3 uses *typed logic*, in the sense that every variable or constant must belong to a type (called a "sort"), and so must also every argument of a function or a predicate. Every function must have a return type; predicates are no more than functions with a boolean return type.

Here is the complete Python program for proving that the formula

$$\exists x \forall y((\neg\mathsf{Shaves}(y, y) \longrightarrow \mathsf{Shaves}(x, y)) \wedge (\mathsf{Shaves}(y, y) \longrightarrow \neg\mathsf{Shaves}(x, y)))$$

is a contradiction:

[3] https://github.com/Z3Prover/z3

```
1  from z3 import *
2
3  Man = DeclareSort('Man')
4  x = Const('x',Man)
5  y = Const('y',Man)
6  Shaves=Function('Shaves',Man,Man,BoolSort())
7
8  s=Solver()
9  s.add(Exists(x,ForAll(y,And(Implies(Not(Shaves(y,y)),Shaves(x,y)),\
10    Implies(Shaves(y,y),Not(Shaves(x,y)))))))
11
12 print(s.check())
```

On line 3, we define the unique sort we need, namely `Man`. On lines 4 and 5, we declare x and y as being of sort `Man`. On line 6, we define a binary function `Shaves` that takes two `Man` arguments and returns a boolean. It is, therefore, a predicate. On line 8, we instantiate our reasoner. It has no formula for the moment. On the next line, we add our formula. Here the Python functions `Exists`, `ForAll`, `And`, `Implies`, `Not`, correspond to symbols $\exists$, $\forall$, $\wedge$, $\rightarrow$, $\neg$. Finally, on line 12, we ask the reasoner whether the formula is satisfiable or not. The answer is firm and definitive:

```
unsat
```

meaning "unsatisfiable," i.e., a contradiction: there is no model for this formula, we can interpret it in any world we may want, it will never be true, and therefore, there will never be a barber with these properties. No barber, no paradox.

10.1.5 Knowledge Bases and Queries

Let us take some (potentially large) set of FOL formulas. Let us build a unique formula by taking their conjunction. A model of this large formula is necessarily also a model of each one of the formulas. We call such a formula a *knowledge base* (KB). For example, there is a formalization of Wikipedia called DBPedia.[4] If we take an interpretation of DBPedia with the real world as domain and with the appropriate choices for each predicate, function, and constant, and if we consider that all formulas are true for this interpretation, then DBPedia is a knowledge base of the real world.

Let KB be a knowledge base of the real world. To prove that some statement α is true in the real world (using the same interpretation as KB), we can start by checking whether α is included in KB.

If KB were a database, we would communicate with it in the SQL language (or in some similar way), querying whether a given information is contained in the database. The result of the query would be either "Yes, here is the information you requested" or "No, this information is unavailable."

A knowledge base is way more powerful than that.

[4] https://www.dbpedia.org/resources/

The power of logic resides in *inference*: if we can infer α from KB, then, thanks to Gödel's completeness theorem, α will be a consequence of KB, in all interpretations, and so whenever our KB is true (which we hope), α will be true as well. It is, therefore, essential to prove that some formula α (which we also call a *query*, like in SQL) can be deduced from a knowledge base.

There are programs like *Z3* in the previous section, called *reasoners*, that can prove or attempt to prove consequences for us, but sometimes these programs are not acting as we would expect. Sometimes they can be pretty counter-intuitive. Therefore, it is good to be able to make some calculations by hand to check out the machine's answers and find flaws in our formulas. Therefore we will quickly present a proof method called *resolution*, used by many reasoners. Knowing how it works also allows a better understanding of the reasoners' debugging output.

10.1.5.1 Resolution

Our goal will be to prove that KB $\models \alpha$. We have the following very nice theorem:

THEOREM. — *We have* KB $\models \alpha$ *iff* KB $\wedge \neg\alpha$ *is a contradiction.*

In other words, to prove that α is a consequence of a knowledge base, take the negation of α and add it to the knowledge base. If the resulting conjunction of formulas is false in all interpretations, i.e., if it is a contradiction, then α is indeed a consequence of KB. This is probably the only theorem proof in this book. Still, we can't resist giving it because it sheds light on the relation between two seemingly very different notions: consequence and contradiction. Here it is:

Proof. — (1) $\Rightarrow$. Suppose we have KB $\models \alpha$, then for every model $M \models$ KB we have $M \models \alpha$, and necessarily $M \not\models \neg\alpha$. We will prove this part by the absurd. Let us suppose that KB $\wedge \neg\alpha$ is not a contradiction. Then there is a $M' \models$ KB $\wedge \neg\alpha$. The model of a conjunction is a model of each part of the conjunction. Therefore, $M' \models$ KB and $M' \models \neg\alpha$. We have just seen that in that case, we must have $M' \not\models \neg\alpha$, which is absurd. Therefore, KB $\wedge \neg\alpha$ is a contradiction.

(2) $\Leftarrow$. If KB $\wedge \neg\alpha$ is a contradiction, then its negation, $\neg(\text{KB} \wedge \neg\alpha) = \neg\text{KB} \vee \alpha = \text{KB} \rightarrow \alpha$ is a tautology. This means that for every $M \models$ KB, we also have $M \models (\text{KB} \rightarrow \alpha)$ (since it is a tautology), and therefore, by modus ponens, $M \models \alpha$, and therefore KB $\models \alpha$. □

As we "only" need to show that KB $\wedge \neg\alpha$ is a contradiction to prove the consequence KB $\models \alpha$, we will simplify our formula so that showing that it is a contradiction becomes easier. We aim to rewrite it as a universally quantified conjunction of smaller formulas, each of these smaller formulas being a disjunction of *literals* (a literal being a predicate or the negation of a predicate). If the formula is a conjunction of smaller formulas, then showing that any subformula is a contradiction is sufficient to prove that the global formula is a contradiction.

To rewrite the formula, three steps are needed: first, we move all quantifiers (universal and existential) to the beginning of the formula, then we replace all connectives with conjunctions and disjunctions, and finally, we remove all existential quantifiers so that only universal quantifiers remain.

To move all quantifiers to the beginning, one must only pay attention to misnamings of variables: in a case like $\exists x \forall y\, P(x, y) \wedge \forall x \exists y\, Q(x, y)$, the x and y variables in the first predicate are not the same as those in the second, as they are in the scopes of different quantifiers. To move the quantifiers of Q to the left, we first rename their variables (after all, they are just dummy variables) into names different than those of P: $\exists x \forall y\, P(x, y) \wedge \forall u \exists v\, Q(u, v)$, and now we can move the quantifier to the left: $\exists x \forall y \exists u \exists v\, P(x, y) \wedge Q(u, v)$. All we need to do is keep the same order of quantifiers.

In the second step, we leave the quantifiers aside and focus on the connectives. Here are four sub-steps that translate any formula into a conjunction of disjunctions of literals (a form called *CNF* = Conjunctive Normal Form):

1. replace double implications by conjunctions of simple implications;
2. replace implications $P \longrightarrow Q$ by $\neg P \vee Q$;
3. enter negations into parentheses (using the formulas $\neg(P \wedge Q) = \neg P \vee \neg Q$ and $\neg(P \vee Q) = \neg P \wedge \neg Q$;
4. use the distributive property to obtain a conjunction of groups of disjunctions.

Here is an example: $B \leftrightarrow (F \vee G)$ becomes $B \longrightarrow (F \vee G) \wedge (F \vee G) \longrightarrow B$ by sub-step 1, then $(\neg B \vee F \vee G) \wedge (\neg(F \vee F) \vee B)$ by sub-step 2, then $(\neg B \vee F \vee G) \wedge ((\neg F \wedge \neg G) \vee B)$ by sub-step 3 and, finally, $(\neg B \vee F \vee G) \wedge (\neg F \vee B) \wedge (\neg G \vee B)$ by sub-step 4. The resulting formula is equivalent to its original form (equivalent in that it has the same models). It is simpler for the machine to process and more difficult for the human to understand.

Until now, the formulas we obtained were strictly equivalent to the original form. To remove the existential quantifiers, we will "lose information." Our third step is called *skolemization*. Let us write our formula as $Q_1 \cdots Q_m \phi(x_1, \ldots, x_m)$ where $x_1, \ldots, x_m$ are the m quantified variables and Q_i are the corresponding quantifiers (universal or existential). If Q_1 is existential, i.e., if it is $\exists x_1$, then there is at least one value for which the formula holds. Let us call this value c (a symbol that does not appear elsewhere in the formula) and replace x_1 with the constant c everywhere in the formula. By this trick, we got rid of $\exists x_1$, but our formula is weaker: indeed, $\exists x_1$ means that there is *at least* one value, so there may be many such values—we have kept only one.

After Q_1 is removed, we do the same with Q_2, and so on, as long as they are existential quantifiers. Imagine now we have a Q_i that is a universal quantifier, and after it, some Q_j is the first existential quantifier: we have $\forall x_i \cdots \forall x_{j-1} \exists x_j$. Then we cannot simply replace x_j by a constant value because its value depends on the values of $x_i, \ldots, x_{j-1}$. We, therefore, replace it with a function $f(x_i, \ldots, x_{j-1})$. The reader should note that this function is not necessarily known or described analytically. All we know is that it exists, so we give it a name. By continuing in the same way, we obtain a formula without any existential quantifier but with some variables replaced by constants or functions of other variables. Here is a generic example: the formula $\exists x \forall y \exists z \forall u\, P(x, y, z, u)$ becomes $\forall y \forall u\, P(c, y, f(y), u)$, where c is a constant. And here is an example where the Skolem function can be calculated: $\forall x \exists y\, x + y = 2$. Its Skolem form is $\forall x\, (x + f(x) = 2)$, and, obviously, $f(x) = 2 - x$.

The reader may have noticed that the formula we obtain after skolemization is not equivalent to the original formula. This is not a problem since our ultimate goal is to show that the formula is a contradiction (remember: our formula is $\mathrm{KB} \wedge \neg\alpha$, and we want to prove it is a contradiction to be able to assert that $\mathrm{KB} \models \alpha$). If the skolemized formula is a contradiction, then the original formula is also a contradiction, and that is all that matters.

Now, our formula is very specific: it contains only universal quantifiers and a conjunction of groups of disjunctions of literals. These groups are called *clauses*, and we will consider them distinct formulas. The *resolution method* is, in fact, an inference rule like modus ponens (in fact, modus ponens is the simplest case of resolution), only it is applied to pairs of clauses. It can be applied only to clauses with a particular property: the first must contain a literal such that the second contains its negation. In other words, if $A_1 \vee \cdots \vee A_n \vee C$ is the first clause, the second must be of the type $B_1 \vee \cdots \vee B_m \vee \neg C$. The inference step is then as follows:

$$\frac{A_1 \vee \cdots \vee A_n \vee C \qquad B_1 \vee \cdots \vee B_m \vee \neg C}{A_1 \vee \cdots \vee A_n \vee B_1 \vee \cdots \vee B_m}\ \text{resolution}$$

The new clause we obtain by inference is called a *resolvent*. We apply resolution inferences to our clauses until we get an empty resolvant. If this is possible, then the original formula is a contradiction. If it is not possible, then it is not a contradiction, but to be able to assert this, we must check out *all* possible combinations of clauses. Before giving an example, let us note that we will also use the rule of universal instantiation to substitute variables with terms or constants.

As an example, let us take, once again, the "All men are mortal" case. Our KB will be

$$\forall X\, \mathsf{man}(X) \longrightarrow \mathsf{mortal}(X)$$
$$\mathsf{man}(\mathsf{Socrates})$$

and our α will be $\mathsf{mortal}(\mathsf{Socrates})$. Therefore $\mathrm{KB} \wedge \neg\alpha$ will be

$$\forall X, (\mathsf{man}(X) \longrightarrow \mathsf{mortal}(X)) \wedge (\mathsf{man}(\mathsf{Socrates})) \wedge (\neg\mathsf{mortal}(\mathsf{Socrates}),$$

and by cleaning this formula up, we get

$$\forall X\ (\neg\mathsf{man}(X) \vee \mathsf{mortal}(X)) \wedge (\mathsf{man}(\mathsf{Socrates})) \wedge (\neg\mathsf{mortal}(\mathsf{Socrates}),$$

that is, three clauses. By universal instantiation, we will replace X by $\mathsf{Socrates}$ so that the first clause has a literal $\neg\mathsf{man}(\mathsf{Socrates})$, which can be eliminated with the unique literal of the second clause. The resolvent will be $\mathsf{mortal}(\mathsf{Socrates})$ which can be eliminated by the negation of our query so that we get an empty resolvent, which proves the contradiction of $\mathrm{KB} \wedge \neg\alpha$ and hence the consequence $\mathrm{KB} \models \alpha$. Here is the complete diagram (S stands for $\mathsf{Socrates}$, to save space):

$$\cfrac{\cfrac{\cfrac{\neg\mathsf{man}(X) \vee \mathsf{mortal}(X)}{\neg\mathsf{man}(S) \vee \mathsf{mortal}(S)}\ \text{univ.inst.} \qquad \mathsf{man}(S)}{\mathsf{mortal}(S)}\ \text{resolution} \qquad \neg\mathsf{mortal}(S)}{\emptyset}\ \text{resolution}$$

Here is the input and output of a reasoner included in the *nltk* distribution: *Prover9*.[5] The program that corresponds to the “All men are mortal“ syllogism is:

```
from nltk.sem import Expression
from nltk.inference import Prover9Command

p1 = Expression.fromstring('all x.(man(x) -> mortal(x))')
p2 = Expression.fromstring('man(socrates)')

query  = Expression.fromstring('mortal(socrates)')
m=Prover9Command(query,[p1,p2])
print(m.prove(verbose=True))
```

The `query` is given separately since it will be negated (it is the $\neg\alpha$). Thanks to the `verbose=True` argument, Prover9 will output debugging code. Here is the part that describes the resolution operation:

```
1  ============================== PROCESS NON-CLAUSAL FORMULAS ==========
2
3  % Formulas that are not ordinary clauses:
4  1 (all x (man(x) -> mortal(x))) # label(non_clause).  [assumption].
5  2 mortal(socrates) # label(non_clause) # label(goal).  [goal].
6
7  ============================== end of process non-clausal formulas ===
8  ============================== PREDICATE ELIMINATION =================
9
10 Eliminating man/1
11 3 man(socrates).  [assumption].
12 4 -man(x) | mortal(x).  [clausify(1)].
13 Derived: mortal(socrates).  [resolve(3,a,4,a)].
14
15 Eliminating mortal/1
16 5 mortal(socrates).  [resolve(3,a,4,a)].
17 6 -mortal(socrates).  [deny(2)].
18 Derived: $F.  [resolve(5,a,6,a)].
19
20 ============================== end predicate elimination =============
```

First, the assumption and the goal (α) are numbered (#1 and #2). Then, to eliminate the `man/1` predicate (the `/1` stands for its arity, so that predicates with the same name but different arities can coexist), on line 10, we have the second assumption, and on line 12, the “clausified” (= disjunction of literals) formula #1. Applying the resolution inference to these two produces the resolvant on line 13. A second inference is applied to this resolvant (line 16) and to the negation of the query (called `deny`); the result is an empty formula (written `$F`).

[5] The “9” is a reference to *Plan 9 from outer space* (1959), the legendary worst-science-fiction-movie-of-all-times in which Bella Lugosi appears for the last time. The biopic *Ed Wood* (1994) relates the making of this movie.

10.2 Extensions of First-Order Logic

> I recall a workshop in which Pat Hayes proposed a fine of $1,000 for every researcher who invents a new logic.
>
> Murray Shanahan, *Solving the frame problem*

We already mentioned that there had been simplifications and extensions of First-Order Logic. In this section, we will go through some extensions of FOL—in Section 10.3, we will turn to a family of fragments of FOL, namely Description Logics.

10.2.1 Temporal Logic: Event Calculus

> Von allem, was der Mensch begehrte, war er immer nur durch Zeit getrennt, nur durch diese Zeit, diese tolle Erfindung![6]
>
> Hermann Hesse, *Klein und Wagner*

There have been many attempts to formalize reasoning about action and time. We will briefly present *event calculus*, a formalism that has *events* and *fluents* (a fluent is like a predicate, but, as its name suggests, it may change over time). Time is considered as a single timeline (definitely not the *wibbly-wobbly timey-wimey thing* the Doctor mentions in *Blink* S29E10). Specifying a set of event occurrences is called a *narrative*. According to Mueller,

> the event calculus supports context-sensitive effects of events, indirect effects, action preconditions, and the commonsense law of inertia. Certain phenomena are addressed more naturally in the event calculus, including concurrent events, continuous time, continuous change, events with duration, nondeterministic effects, partially ordered events, and triggered events. [21, p. 671]

The *commonsense law of inertia* (CLOI) is a rather grand name for a very simple property: if things are not acted upon, they remain unchanged. If you leave this book on your desk and return the day after, it will remain there (provided you have no cat). Event calculus will fill gaps between "snapshots" of a situation: events describe changes in fluent values, and logical inferences allow us to assert what happened in between.

Discrete Event Calculus [20] has a certain number of hardcoded predicates, which we display in Table 10.1. In them, we often see the CLOI. It is crucial since it means that the fluent may evolve from then on, while as long as we are subject to the CLOI, nothing changes. Discrete Event Calculus is based on twelve axioms and definitions, among which are the following:

[6] "For then man was separated from all he craved only by time, by time alone, this crazy invention!"

Table 10.1 Discrete Event Calculus predicates (e is an event, f_* are fluents and t_*, timepoints) [21, p. 679]

Predicate	Meaning
$\mathsf{HoldsAt}(f, t)$	f is true at time t
$\mathsf{Happens}(e, t)$	e occurs at time t
$\mathsf{ReleasedAt}(f, t)$	fluent f is released from CLOI at t
$\mathsf{Initiates}(e, f, t)$	if e occurs at t, then f is true and not released from the commonsense law of inertia after t
$\mathsf{Terminates}(e, f, t)$	if e occurs at t, then f is false and not released from CLOI after t
$\mathsf{Releases}(e, f, t)$	if e occurs at t, then f is released from CLOI after t
$\mathsf{Trajectory}(f_1, t_1, f_2, t_2)$	if f_1 is initiated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$
$\mathsf{AntiTrajectory}(f_1, t_1, f_2, t_2)$	if f_1 is terminated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$

DEC1 $\mathsf{StoppedIn}(t_1, f, t_2) := \exists e, t\,(\mathsf{Happens}(e, t) \wedge t_1 < t < t_2 \wedge \mathsf{Terminates}(e, f, t))$, which defines the fact that the fluent f is stopped somewhere between t_1 and t_2 as some event e occurred at some time t, where $t_1 < t < t_2$, and at that time, because of that event, the fluent became false, while remaining subject to CLOI;

DEC3 $\mathsf{Happens}(e, t_1) \wedge \mathsf{Initiates}(e, f_1, t_2) \wedge 0 < t_2 \wedge \mathsf{Trajectory}(f_1, t_1, f_2, t_2) \wedge \neg\mathsf{StoppedIn}(t_1, f_1, t_1 + t_2) \longrightarrow \mathsf{HoldsAt}(f_2, t_1 + t_2)$.
This axiom says that when an event happens at t_1 and thereby sets f_1 to true (while staying under CLOI). There is a "trajectory," i.e., a link between f_1 and f_2 such that if f_1 is initiated at t_1, then f_2 is still true at $t_1 + t_2$, and, finally, f_1 has not been stopped (in the sense of DEC1) between t_1 and $t_1 + t_2$—if all these conditions are met, then we have that f_2 is true at time $t_1 + t_2$. In other words, we can rely on fluents not changing when there is no reason for them to change;

DEC6 $\neg\mathsf{HoldsAt}(f, t) \wedge \neg\mathsf{ReleasedAt}(f, t+1) \wedge \neg\exists e(\mathsf{Happens}(e, t) \wedge \mathsf{Initiates}(e, f, t)) \longrightarrow \neg\mathsf{HoldsAt}(f, t + 1)$, i.e., if f is false at time t and is not released from CLOI at $t + 1$ and if there happens no event at t that would set f true, then at $t + 1$ (time is discrete) we are still in the same situation: f is still false.

Mueller [21] gives the following example to illustrate the formalism: the act of turning on and off a light. We have two event classes e: TurnOn and $\mathsf{TurnOff}$, and a fluent f taking two values: On and Off. Here is the description of the event:

$$\mathsf{Initiates}(e, f, t) := (e = \mathsf{TurnOn} \wedge f = \mathsf{On}); \tag{10.1}$$

$$\mathsf{Terminates}(e, f, t) := (e = \mathsf{TurnOff} \wedge f = \mathsf{Off}); \tag{10.2}$$

$$\mathsf{Happens}(e, t) := (e = \mathsf{TurnOn} \wedge t = 2) \vee (e = \mathsf{TurnOff} \wedge t = 4); \tag{10.3}$$

$$\mathsf{TurnOn} \neq \mathsf{TurnOff}; \tag{10.4}$$

$$\neg\mathsf{Releases}(e, f, t); \tag{10.5}$$

$$\neg\mathsf{ReleasedAt}(f, t);\ \text{(at no time does someone act upon the light switch)} \tag{10.6}$$

$$\neg\mathsf{HoldsAt}(\mathsf{On}, 0). \tag{10.7}$$

In other words, we have two events: light is turned on at time 2 and turned off at time 4. $\mathtt{TurnOn}$ and $\mathtt{TurnOff}$ are not the same event (in logic, one must be careful: constants may carry different names but be sent to the same domain element when interpreted; the inequality relation guarantees that this will not happen, whatever the interpretation). None of the two events releases the fluent from CLOI, and it isn't released for any other reason (i.e., there is no telepath in the room, nor is the light switch a living entity), and finally, before turning the light on, it was off.

After having modeled the situation, Mueller [21, pp. 682–683] shows that three formulas are consequences of the formulas above and the twelve axioms: $\neg\mathtt{Holds}(\mathtt{On}, 1)$ (the light is still off at $t = 1$), $\mathtt{Holds}(\mathtt{On}, 3)$ (the light is on at $t = 3$) and $\neg\mathtt{Holds}(\mathtt{On}, 5)$ (it is again off at $t = 5$). Let us show the first one: (10.3) tells us that events occur only at $t = 2$ and $t = 4$. Therefore, at $t = 0$ nothing happened, and, in particular, nothing was initiated: $\neg\exists e(\mathtt{Happens}(e, 0) \wedge \mathtt{Initiates}(e, \mathtt{On}, 0)$. This is already part of the left side of DEC6 (applied to $t = 0$). From (10.7) (for $t = 0$) we have that $\neg\mathtt{HoldsAt}(\mathtt{On}, 0)$, and from (10.6) (for $t = 1$) we have $\neg\mathtt{ReleasedAt}(\mathtt{On}, 1)$. Now we have the complete left part of DEC6, which implies the right part, namely $\neg\mathtt{HoldsAt}(\mathtt{On}, 1)$, and we are done.

10.2.2 Modal Logic

Modal logic is a framework introduced by the (recently deceased) philosopher Kripke in the late fifties (when he was still a teenager!), allowing the definition of various logics applied to specific situations. This framework considers adding a *modal operator* $\Box$ to Propositional Logic and a special way of interpreting the formula.

Here is how it works. When we discussed Propositional Logic, we said that the domain of an interpretation could finally be arbitrary since the only symbols we have are propositions. Their interpretations are no more than truth values. Here, we will do as if 0-arity predicates were 1-arity predicates so that there is a set of elements of the domain for which they are true. Imagine we take the real world as the domain and consider the single-argument predicate $\mathtt{mortal}()$. It is true for many entities of the real world (all (wo)men, mammals, birds, etc.) but not for all of them (inanimate entities are not mortal, and neither is our beloved t.dohrnii). The set of elements of the domain for which the predicate is true can be considered the interpretation of $\mathtt{mortal}()$.

Modal logic proceeds similarly with propositions: it considers that the interpretation of a proposition is some subset of the domain. We define a *pointwise model* as follows: let $M = (W, R, \mathbf{I})$ be a model, where for a proposition P, $P^{\mathbf{I}}$ is a subset of W; we have $M, w \models P$ iff $w \in P^{\mathbf{I}}$, and we define inductively the pointwise model of negations, conjunctions, disjunctions, and implications. By this definition, we have three levels of satisfaction: a formula ϕ is *satisfiable in* M if there is a $w \in W$ such that $M, w \models \phi$; it is *globally true* if $M, w \models \phi$ for all $w \in W$; and it is *valid* (a *tautology*) if it is true for all elements of all models.

Let us now define the operator $\Box$: we have $M, w \models \Box\phi$ iff we have $M, w' \models \phi$ for *all* outgoing neighbors of w for the binary relation R. That is, we take the set of all neighbors of w in the digraph of R, and $\Box\phi$ is true at w iff ϕ is true for all of them. In particular, if w has no neighbors for the relation R, then $\Box\phi$ is true at w. In Fig. 10.1, taken from Blackburn *et al.* [4, p. 8], we display such a binary relation R as a digraph. Nodes 1 and 2 are those where ϕ is true. As for $\Box\phi$, it will be true at 1 if all its outgoing neighbors are true for ϕ, this is not the case (neither 3 nor 4 are true for ϕ). But $\Box\phi$ is true at 2 since its only outgoing neighbor, namely 1, is true for ϕ. But $\Box\phi$ is not true at 4 since it has two outgoing neighbors: 2 and itself, which is not true for ϕ. Finally, $\Box\phi$ is true at 3 since 3 has no outgoing neighbors.

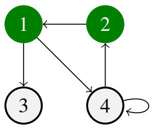


Fig. 10.1 Relations between elements of W. Edges denote the binary relation R. ϕ is true on green nodes. [4, p. 8]

We define a second operator, $\Diamond$ as $\Diamond\phi := \neg\Box\neg\phi$. By definition, $\Diamond\phi$ will be true at w if *at least one* neighbor of w is true for ϕ. In Fig. 10.1, this is the case for 2 and also for 4 (since it has at least one true outgoing neighbor, namely 2), but it is not true for 3 (at least one outgoing neighbor is required, and 3 has none).

Blackburn *et al.* use Fig. 10.1 as an illustration of the calculation of formula $\Diamond\Box\Diamond\phi$. They claim that this formula is true at 1 but not at 2. Let us verify this claim: we first take some neighbors of 1, for example, $\{4\}$, then we take all their neighbors: $\{2{,}4\}$, and finally, at least one outgoing neighbor of each one of them: $\{1{,}2\}$, these are true and so $\Diamond\Box\Diamond\phi$ is true at 1. As for 2, if we take at least one neighbor, it has to be $\{1\}$, then we take its complete set of neighbors: $\{3{,}4\}$, and finally, we need at least one neighbor for each one of this, and this is impossible for 3, which has no neighbor at all. Therefore, $\Diamond\Box\Diamond\phi$ is false at 2.

This is the general scheme of modal logics. The reader may wonder what it is suitable for. After all, it is definitely abstract and does not explain what the binary relation R may be. In the following, we will consider three particular FOL extensions, all with fancy Greek names: *alethic logic*, *epistemic logic*, and *deontic logic*, and the reader will see that they are just applications of modal logic to specific contexts. Even *description logics*, which we will cover afterward, is, in fact, a disguised modal logic as shown by Schild [31].

10.2.2.1 Alethic Logic

The word "alethic" comes from the Greek "ἀλήθεια"$_{\text{EL}}$ ⌊truth⌋, in the sense that it deals with the "modalities of truth," namely logical necessity, contingency, possibility, and impossibility. In this specific logic, the operator $\Box$ has the meaning "it is necessary that," so that $\Box\neg P$ means that it is necessary that P be false, i.e., P is impossible, and, logically, $\Diamond P = \neg\Box\neg P$ means that P is possible.

One way to describe W is as the "set of possible worlds," so that each $w \in W$ is a "world," and if $M, w \models P$, then P is true in that world. The binary relation R becomes then an *accessibility relation* between the worlds, and the definition of $\Box$ becomes: $M, w \models \Box P$ means that P is true in all worlds accessible from w. Intuitively, this means that when we say that worlds are "accessible," then they are similar enough, and $\Box P$ means that P is true in all neighbors of some world. And being possible $\Diamond P$ means that P may not be true in all neighbors of w but that P is true in at least one other world. Alethic logic has two additional axioms: $\Box(P \wedge Q) \leftrightarrow (\Box P \wedge \Box Q)$ (necessity of conjunction and conjunction of necessities have the same truth values) and $\Box\top$ (the domain is necessary). It also has an additional inference rule that takes up the axiom:

$$\frac{P \leftrightarrow Q}{\Box P \leftrightarrow \Box Q}\ \text{equivalence}$$

from which, using modus ponens, we can derive the *monotonicity inference rule*:

$$\frac{P \rightarrow Q}{\Box P \rightarrow \Box Q}\ \text{equivalence}$$

10.2.2.2 Epistemic Logic

The word "epistemic" comes from the Greek "ἐπιστήμη"$_{\text{EL}}$ ⌊science⌋ (as in "epistemology"), in the sense of *knowledge*. The operator $\Box$ means "it is known that," and it is most often written "K" (for "knowledge," but also in honor of Kripke). In the context of multi-agents, one will have K_i for every agent, so that $K_i P$ means: "agent i knows P". The axioms of epistemic logic are an extension of those of alethic logic: $KP \rightarrow P$ (when something is known, then it is true. This axiom makes the distinction between (scientific) knowledge and common beliefs or rumors, which can be false), $KP \rightarrow KKP$ (when I know something, I know that I know it), and $\neg KP \rightarrow K\neg KP$ (when I don't know something, then I know that I don't know it). Of course, in real life, these axioms are hardly true, but they are essential to make reasoning possible.

It can be shown that we have an "epistemic modus ponens": $\wedge K(P \rightarrow Q) \vdash KQ$.

The accessibility relation, in this case, is connecting a world to "epistemic alternatives of it." It is an equivalence: the agent cannot distinguish accessible worlds due to her lack of knowledge [1, p. 994].

Let us solve a puzzle, the "Two Muddy Children Problem," taken from Perkowski [25]. It goes as follows:

> There are n children meeting their father after playing in the mud. The father notices that k among the children have mud dots on their foreheads. Each child sees everybody else's foreheads but not his own. The father says: "At least one of you has mud on his forehead."

> The father then says: "Do any of you know that you have mud on your forehead? If you do, raise your hand now." No one raises his hand. The father repeats the question, and again, no one moves. After exactly k repetitions, all children with muddy foreheads raise their hands simultaneously. [25, p. 99]

If we take $n = 2$, in which case the father asserts that *one* child has mud on his head, it is after *one* repetition that the first child will know for sure that it has mud on his head. We will prove this fact using epistemic logic. Here are some facts:

> (1) A and B know that each can see the other's forehead. Thus, for example:
> (1a) If A does not have a muddy spot, B will know that A does not have a muddy spot.
> (1b) A knows (1a).
> (2) A and B each know that at least one of them has a muddy spot, and they each know that the other knows that. In particular,
> (2a) A knows that B knows that either A or B has a muddy spot.
> (3) B says that he does not know whether he has a muddy spot, and A thereby knows that B does not know.

out of which we will use (1b), (2a) and (3). Furthermore, we will use two axioms

- (ax1): $K(\phi) \wedge K(\phi \to \psi) \to K(\psi)$;
- (ax2): $K(\phi) \to \phi$,

and the following inference rules:

- standard modus ponens (m.p.): $P,\ P \to Q \vdash Q$;
- logical omniscience (l.o.): $KP,\ P \to Q \vdash KQ$, this is a weaker version of the epistemic modus ponens;
- material implication (m.i.): $P \vee Q \vdash \neg P \to Q$ (we have presented this as the very definition of implication, but it has a particular name because in some other logics, such as *intuitionistic logic*, it is not accepted);
- monotonicity (mon.): $Q,\ P \wedge Q \to R \vdash P \to R$. This rule comes from the fact that the second premise is $\neg P \vee \neg Q \neg R$, and therefore if we know that we have Q, we can remove $\neg Q$ from the disjunction, and the result is $\neg P \vee R$, which by material implication becomes $P \to R$;
- transitivity (tr.): $P \to Q,\ Q \to R \vdash P \to R$;
- contrapositivity (con.): $P \to Q \vdash \neg Q \to \neg P$.

Here is the proof. We have broken it into two parts to make it fit on the printed page. We will call the first part "RIGHT_SIDE":

$$\dfrac{\dfrac{\dfrac{(2a)\ K_A(K_B(\mathsf{md}(A) \vee \mathsf{md}(B))) \qquad (ax2)\ K\phi \to \phi}{K_B(\mathsf{md}(A) \vee \mathsf{md}(B))}\ \text{m.p.}}{K_B(\neg\mathsf{md}(A) \to \mathsf{md}(B))}\ \text{m.i.} \qquad (ax1)\ K(\phi) \wedge K(\phi \to \psi) \to K(\psi)}{K_B(\neg\mathsf{md}(A)) \to K_B(\mathsf{md}(B))}\ \text{mon.}$$

The second part follows:

$$\dfrac{(3)\ K_A(\neg K_B(\mathsf{md}(B))) \qquad \dfrac{\dfrac{\dfrac{(1b)\ K_A(\neg\mathsf{md}(A)\to K_B(\neg\mathsf{md}(A))) \qquad (ax2)\ K\phi\to\phi}{\neg\mathsf{md}(A)\to K_B(\neg\mathsf{md}(A))}\,\text{m.p.} \qquad \text{RIGHT\_SIDE}}{\neg\mathsf{md}(A)\to K_B(\mathsf{md}(B))}\,\text{tr.}}{\neg K_B(\mathsf{md}(B))\to\mathsf{md}(A)}\,\text{con.}}{K_A(\mathsf{md}(A))}\,\text{l.o.}$$

Epistemic logic has applications in the design of intelligent systems, e-learning, cybersecurity, and game theory.

10.2.2.3 Deontic Logic

The word "deontic" comes from the Greek "δέον"$_{\text{EL}}$ ⌊appropriate to do⌋ (as in "deontology"). There are three modal operators: OP (P is obligatory), FP (P is forbidden, equivalent to O$\neg P$ = it is mandatory not to do P), and PP (P is permitted, equivalent to $\neg$O$\neg P$ = it is not mandatory not to do P, i.e., it is allowed to do P). From those, one can define more operators, such as "optional" ($\neg$O$P \wedge \neg$OP: P is neither mandatory nor forbidden), "non-optional" (O$P \vee$ O$\neg P$, either it is mandatory to do P, or it is mandatory not to do P, i.e., in both cases, one does not have the choice).

The accessibility relation points to the "ideal" or "perfect deontic alternatives" of the world under consideration. The crux behind this is that in some possible world, something (say ϕ) is mandatory if ϕ holds in all the perfect alternatives of this world, as indicated by the accessibility relation [1, p. 996].

Deontic logic has applications in writing and assessing regulations, laws, contracts, etc., by humans or machines.

10.2.2.4 Other Modal Logics

Besides alethic, epistemic and deontic logic, there is also a *temporal* logic (where R is simply the fact of being later in the timeline), a *doxastic* logic ($\Box$ becomes "it is believed that"), an *ethical* logic ($\Box$ becomes "It is good that"), etc.

10.3 Description Logics

Minsky's frames are a pioneering work in Artificial Intelligence. They were intended ☞ 159
for knowledge representation, but it quickly became clear that they lacked a formal substrate, such as some (formal) logic. This motivated the introduction of *Description Logics* in the 1980s. As the goals of Description Logics were mainly computational, they have been designed modularly to optimize the ratio between expressivity and processing requirements. Each ingredient can be included individually so that one has not one but a whole family of logics named by the initials of the

various ingredients: $\mathcal{AL}$ (polynomial complexity), $\mathcal{ALE}$ (NP-complete), $\mathcal{ALCN}$ (PSpace-complete), etc. [1, p. 776].

Description Logics are not modal because they have some modal operator with specific semantics. Still, they take over Kripke's idea of using a binary relation and defining $\Box$ and $\Diamond$ for this binary relation. The idea is generalized to an arbitrary number of binary relations called *roles*, and for a given role r and a set of elements of the domain C, $\Box C$ is written $\forall r.C$ (the set of all elements that have *all* their neighbors for the r binary relation in C, or those who have no r-neighbors), and $\Diamond C$ is written $\exists r.C$ (the set of all elements that have *at least one* r-neighbor in C). So, although description logics are essentially not modal, they have the syntax of a modal language, as shown by Schild [31]. Therefore, all algorithms and methods developed for modal languages have been trivially adapted to description logics.

10.3.1 Definitions

The basic elements of Description Logics (DL) are *individuals*. Individuals belong to *classes* and participate in binary relations called *roles*. In a DL knowledge base, one has three kinds of statements:

1. those "belonging to the TBox" ("T" as in "terminology") which are assertions about individuals belonging to classes—in FOL, they would be unary predicates;
2. those "belonging to the ABox" ("A" as in "assertions") which are assertions about set-theoretical relations between classes—in FOL, they would be implications between unary predicates;
3. those "belonging to the RBox" ("R" as in "roles") which are assertions about set-theoretical relations between roles—in FOL, they would be implications between binary predicates.

A *class* is a set of individuals. In FOL, this corresponds to the interpretation of a unary predicate: the set of elements of the domain for which the predicate is true. A *role* is a set of binary relations between individuals. In FOL, this corresponds to ordered pairs of individuals for which the interpretation of a binary predicate is true. The entire domain is called *universal class*, and written $\top$. The empty set is called *empty class* and written $\bot$.

If C_1 and C_2 are classes, their intersection is written as $C_1 \sqcap C_2$. In FOL, this corresponds to the interpretation of $C_1(X) \wedge C_2(X)$. Similarly, if C_1 and C_2 are classes, their union is written $C_1 \sqcup C_2$, which in FOL corresponds to the interpretation of $C_1(X) \vee C_2(X)$. If C_1 is a class, its complementary set $\neg C_1$ corresponds in FOL to the negation $\neg C_1(X)$.

Now we turn to notations that are specific to DL. If R is a role and C a class, the *existential restriction* $\exists R.C$ is the set of elements x such that we have at least one $R(x, y)$, with y in C (we recognize the $\Diamond$ operator from modal logic). As an example, let us consider the role "being a father of" and the class of "girls," then $\exists$father_of.woman," are those individuals that are fathers of one girl or more.

Similarly, if R is a role and C a class, the *universal restriction* $\forall R.C$ is the set of elements x such that

1. either all y such that we have $R(x, y)$ are in C, or
2. there is no y such that we have $R(x, y)$

(we recognize the $\Box$ operator of modal logic). Taking the previous example again, ∀father_of.woman will be the individuals that are fathers of girls only, or not fathers at all.

Imagine you wish to avoid the case where there is no successor for a given role (as in the previous example, people not being fathers). It suffices to take the intersection $\forall R.C \sqcap \exists R.C$ (the individuals, all children of which are girls and having at least one child, therefore necessarily a girl).

The notations $\exists R.C$ and $\forall R.C$ are to be distinguished from the regular use of $\forall$ and $\exists$ in mathematics or io FOL. Indeed, in mathematics or FOL, the quantifier is followed by a variable, the values of which are subject to the constraint expressed by the quantifier. In the case of DL, there are no variables, and the quantifier is followed by the role, which is the specific binary relation we are interested in. After a dot follows the target class. The awkwardness of this notation resides in the fact that the individuals in this set are not explicitly represented. Their set is implicitly described by the binary relation of which they are predecessors and the container of the corresponding successors. C can also be $\top$ or $\bot$:

- $\forall R.\top$ will be
  - the set of all x for which all y such that we have $R(x, y)$ are in $\top$, but this is the case of every y, and therefore of all x that participate to the relation R, and
  - the set of all x for which there is no y such that we have $R(x, y)$. In other words, $\forall R.\top$ is necessarily $\top$;
- $\forall R.\bot$ is the set of the x for which all y such that $R(x, y)$ belong to the empty class. There is no such y and therefore $\forall R.\bot$ is the set of x that do *not* participate in a R binary relation;
- $\exists R.\top$ is the set of all x that participate in at least one binary relation R, with no restriction on y;
- $\exists R.\bot$ is the set of all x that participate in at least one binary relation $R(x, y)$, with y belonging to the empty set, this cannot be, so that $\exists R.\bot$ is necessarily $\bot$ (which is symmetric with $\forall R.\top$).

Once a KB in DL has been built, one may want to ask questions as the following [16, p. 61]:

- is the KB *consistent*? This amounts to proving that the KB is satisfiable;
- is a given class or role *satisfiable*? I.e., can this class or role be non-empty without leading to contradictions?
- class or role *subsumption*: do we have $C_1 \sqsubset C_2$ or $R_1 \sqsubset R_2$ for some classes C_1, C_2, or roles R_1, R_2?

- *instance checking*: is some individual a class member? Is a given pair of individuals a member of a given relation?
- *instance retrieval*: find all members of a given class or relation.

10.3.2 An Example: DL Formulas for AWO

In the next chapter, we will explore ontologies, that is, knowledge bases using DL formulas for describing a knowledge domain. The example of ontology we will use will be AWO (African Wildlife Ontology), a tutorial ontology introduced in 2010 [17, 16].

AWO aims to describe the classes of animals, plants, lions, giraffes, etc. Let us translate the following sentences into DL: "Plants are disjoint from animals," "Branches are parts of trees," "Leaves are parts of branches," "Carnivores are exactly those animals that eat only animals," "Herbivores are exactly those animals that eat only plants or parts of plants," "Lions are animals that eat only herbivores," "Giraffes eat only leaves."

Plants are disjoint from animals

We will use classes plants and animals. Disjointness can be obtained by requiring an empty intersection:

$$\mathsf{plants} \sqcap \mathsf{animals} \equiv \bot.$$

Branches are parts of trees, leaves are parts of branches

We will use classes leaves, branches, and trees. For the sake of completion, we will also translate the fact that trees are plants:

$$\begin{aligned}\mathsf{branches} &\sqsubset \exists\mathsf{is\text{-}part\text{-}of.trees}\\ \mathsf{leaves} &\sqsubset \exists\mathsf{is\text{-}part\text{-}of.branches}\\ \mathsf{trees} &\sqsubset \mathsf{plants},\end{aligned}$$

where is-part-of is a role. We use it as follows: branches are parts of trees in the sense that for every branch b, there exists at least one tree t such that b is part of t. The $\sqsubset$ relations imply that trees may have other parts than branches, and branches may have other parts than leaves.

Carnivores are exactly those animals that eat only animals

We will define class $\mathsf{Carnivores}$ as the subclass of animals that eat only animals. By using the universal quantifier on the role eats, we obtain animals that eat only animals but also animals that do not eat at all. From a Darwinian point of view, a species of anorexic animals would be extinguished in no time. Still, for the sake of completeness, we will include in our formula the fact that we want only animals that indeed eat something:

$$\mathsf{carnivores} \equiv \forall \mathsf{eats}.\mathsf{animals} \sqcap \exists \mathsf{eats}.\top.$$

Herbivores are exactly those animals that eat only plants or parts of plants

$$\mathsf{herbivores} \equiv \forall \mathsf{eats}.(\mathsf{plants} \sqcup \exists \mathsf{is\text{-}part\text{-}of}.\mathsf{plants}) \sqcap \exists \mathsf{eats}.\top.$$

This formula very nicely shows the fact that quantifiers can be used anywhere. Indeed, the subsets we build can be used directly in the formula without being named.

Lions are animals that eat only herbivores, giraffes eat only leaves

These formulas are straightforward variations of the preceding ones:

$$\mathsf{lions} \sqsubset \forall \mathsf{eats}.\mathsf{herbivores} \sqcap \exists \mathsf{eats}.\top,$$
$$\mathsf{giraffes} \sqsubset \forall \mathsf{eats}.\mathsf{leaves} \sqcap \exists \mathsf{eats}.\top.$$

10.3.3 Role Properties

Here are some more tools that can be useful when writing DL formulas. We can define the *inverse* of a role R by $R^-(a, b) := R(b, a)$. Using this notation, a role R is *symmetric* when $R \equiv R^-$.

A role R is *functional* when for each x there is, at most, a single y such that $R(x, y)$ (this corresponds to the definition of function in mathematics: we may take this for granted, but the characteristic property of all mathematical functions is that they send each value of their domain to a single value of the codomain, this is why sin is a function, but sin^{-1} is not a function unless we restrict it to an interval of width less than 2π). A role R is *inverse functional* (like sin^{-1}) when R^- is functional.

A role R is *transitive* when for all x, y, z such that $R(x, y)$ and $R(y, z)$, we automatically have $R(x, z)$. Sometimes we want a role to be transitive, even if this amounts to adding some extra pairs, therefore, we define the *transitive closure* $\overline{R}$ or a role R as the smallest transitive role containing R (for a mathematician, saying that an “application contains another application” may seem weird, but, it suffices to consider an application as a binary relation, that is a set of pairs of elements—

inclusions, intersections, and all other set-theoretic operations are based on these sets of pairs). It is easy to show that R is transitive iff $R \equiv \overline{R}$. A typical transitive role is the one of ancestor in the context of genealogy. The role of parent is strictly intransitive since the parent of a parent is not a parent (but a grandparent), except, of course, in the case of the movie *Predestination* (§ 10.4.3.2).

A role R is *symmetric* if whenever we have $R(x, y)$, we also have $R(y, x)$. Friendship (at least as defined in social networks such as Facebook) is a symmetric relation. A role R is *asymmetric* if one cannot have $R(y, x)$ and $R(x, y)$ simultaneously. The "being a parent of" relation is a typical example of asymmetric relation, except, of course, in the case of the movie *Predestination* (§ 10.4.3.2).

A role R is *reflexive* if we have $R(x, x)$ for all x, as it happens in a mirror where one sees one's reflection. Equality is a typical reflexive relation. And as reflexivity is connected to *fixed point* problems (a fixed point of f is an x such that $f(x) = x$) such as Brouwer's Fixed Point Theorem stating that any continuous mapping of the sphere into itself must have a fixed point (or maps some point to its antipodal), it is also important to study "partial reflexivity," i.e., to identify the set of elements that relate to themselves for a given role R. A special notation for this is $\exists R.\mathsf{Self}$, meaning the *reflexivity set* of R. For example, the set of people shaving themselves would be $\exists\mathsf{Shave}.\mathsf{Self}$ and therefore, the formula of the Barber's Paradox (cf. § 10.1.4.1) in DL would be

$$\neg\exists\mathsf{Shave}.\mathsf{Self} \equiv \exists\mathsf{Shave}^{-1}.\mathsf{Barbers},$$

where $\mathsf{Barbers}$ is the set of barbers. Some explanations: on the left side of the equivalence, we have the set of those that do not shave themselves—on the right side of the equivalence, we have those that at least one barber shaves. The equivalence means that those whom a barber shaves are exactly the non-auto shavers. It is easy to verify that $\mathsf{Barbers}$ must be empty for this formula to be true.

10.3.4 A Naming Scheme for Description Logics

This section will describe and name various family members of description logics. They are all named by abbreviations—the convention is to typeset these abbreviations using math capitals despite the awkwardness of the result. The basic description logic type is called $\mathcal{AL}$, "Attributive Language." The ingredients of $\mathcal{AL}$ are class and role names, $\top$, $\bot$, $\sqcap$, and $\forall$. All the AWO formulas of the preceding section are valid $\mathcal{AL}$ formulas.

To go beyond $\mathcal{AL}$, we have the following modules [2, p. 497]:

- $\mathcal{C}$: class negation;
- $\mathcal{U}$: union (if you have class negation, then you can obtain union as the negation of the intersection of negations);
- $\mathcal{E}$: existential quantifier (if you have class negation, you can obtain the existential quantifier by the negation of the universal quantifier of the negation of the role);
- $\mathcal{F}$: functional properties by using $\leq 1R.C$: a role R is functional if $\exists R.C \sqsubseteq\leq 1R.C$;

- $\mathcal{R}$: role inclusion, reflexivity, irreflexivity, role exclusion;
- $\mathcal{R}^+$: same as $\mathcal{R}$ but with transitive closure;
- $\mathcal{H}$: role hierarchy;
- $\mathcal{O}$: the possibility of describing a class extensionally, that is by an enumeration of its members;
- $\mathcal{I}$: the possibility of having inverse roles (R^-);
- $\mathcal{Q}$: the possibility of having fully quantified restrictions ($\leq n$, $\geq n$, $= n$).

Notice that the letter $\mathcal{S}$ stands as an abbreviation of the abbreviation $\mathcal{ALCR}^+$. Common DL types are $\mathcal{ALC}$, $\mathcal{SHIQ}$, $\mathcal{SHOIN}$ (used by the *Protégé* ontology editor and by OWL-DL),

The formula for the Barber's Paradox requires reflexivity and an inverse role. Therefore, it needs at least $\mathcal{ALIR}$.

10.4 Further Reading

10.4.1 Literature

Textbooks on logic address various audiences. There are

- those targeting computer scientists, such as Ben Ari [3], which covers propositional logic and FOL in a very thorough but always accessible way, presenting many techniques and giving theorem proofs only when necessary; also Lalement [19], starting with terms and λ-calculus in full abstraction, before moving to FOL, model theory and finally computability issues, a masterwork of French precision translated into English; or Harrison [14], who implements everything in the OCaml programming language;
- those targeting mathematicians/logicians, such as Hedman [15], which after presenting FOL, deals with first-order theories, countable theories, and finally Gödel's incompleteness theorems, but always in a very accessible and reader-friendly way; or Ebbinghaus *et al.* [10], which can be used as an advanced panorama of logic and proof theory for mathematicians;
- those targeting linguists, like Gamut [13, 12] (Gamut is actually a collective pseudonym) that covers propositional logic, FOL, intensional logic (as employed by Montague), modal logics, etc.

A very interesting book we highly recommend is Kowalski [18], with the subtitle "How to be Artificially Intelligent." It gives insight into the relations between AI, natural language, logic, and human rationality and is always a pleasant read. From a different point of view, Novaes [23] also studies the cognitive impact of formal languages on human reasoning.

Readers of French can read the introductory texts by Delahaye [8, 7] and his textbook [9], one of the best textbooks on logic ever written.

10.4.2 LaTeX

The *bussproofs* package and its extension *bussproofs-extra* allow typesetting of proofs using a logical approach. For every inference step, one will use commands `\AxiomC` for each formula of the premise a `\RightLabel` command for the inference method's name and commands `\UnaryInfC`, `\BinaryInfC`, `\TernaryInfC`, `\QuaternaryInfC`, etc. for the conclusion of the inference, depending on the number of arguments of the premise. For example, the proof tree

$$\dfrac{\dfrac{\forall X\,\mathrm{man}(X)\to\mathrm{mortal}(X)}{\mathrm{man}(\mathrm{Socrates})\to\mathrm{mortal}(\mathrm{Socrates})}\ \text{univ. inst.}\qquad\quad \mathrm{man}(\mathrm{Socrates})}{\mathrm{mortal}(\mathrm{Socrates})}\ \text{modus ponens}$$

is obtained by the code

```
1 \begin{prooftree}
2 \AxiomC{$\forall X\,\mathrm{man}(X)\to\mathrm{mortal}(X)$}
3 \RightLabel{univ. inst.}
4 \UnaryInfC{$\mathrm{man}(\mathrm{Socrates})\to%
5 \mathrm{mortal}(\mathrm{Socrates})$}
6 \AxiomC{$\mathrm{man}(\mathrm{Socrates})$}
7 \RightLabel{modus ponens}
8 \BinaryInfC{$\mathrm{mortal}(\mathrm{Socrates})$}
9 \end{prooftree}
```

where the command of Line 4 produces an inference step, the conclusion of which is used as a premise argument for the inference on Line 8.

10.4.3 Science Fiction

10.4.3.1 Mr Spock

We all know Mr. Spock, the green-blooded, pointy-eared Vulcan alien in Star Trek. In the early sixties, the trio Captain Kirk / Dr. McCoy / Mr. Spock played the role of a three-fold behavioral and ethical model for the American youth: Dr. McCoy was emotional and had a strong character (and very often saw himself as an old-style village doctor), Mr. Spock was “logical” and supposedly emotionless, and Captain Kirk was a man of action and bravery. Teenagers, in this tumultuous period of life, adolescence, could identify themselves to either of the three characters; there were always the other two to establish an equilibrium.

“Logic” was undoubtedly part of the (classic) Star Trek vocabulary: in three years (80 episodes), the word “illogical” is uttered 62 times, always by Spock. The words “logical” or “logically” are uttered 125 times. By way of comparison, in the Whovian world, the term “illogical” is uttered only 18 times, and “logical(ly)” 90 times in 39 years (874 episodes).

But was it logic that characterized Mr. Spock? As we have seen in this chapter, logic is mostly about representing knowledge and reasoning. Spock may sometimes have made proposals to solve problems in ways that seemed to lack empathy, but by no means is the lack of empathy reasoning. Probably, Gene Roddenberry didn't want a particular character to be the "Enterprise's Sherlock Holmes" to avoid undermining Krik's authority. Spock's "illogical" comments mainly addressed human weaknesses or paradoxes of human behavior.

If correct predictions can evaluate the quality of reasoning, then, as Galef [11] points out, Spock didn't excel in reasoning either. As she says:

> "There's only a very slight chance this will work," Spock warns Kirk in one episode of the original TV show [*Friday's Child* S2E11], right before their plan works. The odds of survival are "less than seven thousand to one," Spock tells Kirk in another episode [*Errand of Mercy* S1E26], shortly before they escape unharmed. The chance of finding survivors is "absolutely none," Spock declares in yet another episode [*This Side of Paradise* S1E24], right before they discover a large colony of survivors.

What characterizes Spock is that he was *different*. As the only alien on board the Enterprise (the staff of which was multicultural but otherwise human), he was supposed to have green blood, but his blood was never seen in the series. His only apparent physical difference is his pointed years. As many authors have noticed, and as he wrote himself in his autobiography [22], Spock was the designated "other," Star Trek's wandering Jew, and this is consistent with his personal experience:

> Who among us does not understand what it is to be an outsider, separate? Even at that tender age, I did. I was a Jewish kid living in a mostly Italian neighborhood. Many of my close friends were Italian, but I learned early on that I was somehow "different" from them. Our friendships stopped at the church door. [22]

Nimoy's parents came from a pogrom-plagued Ukrainian Jewish village; his mother tongue was Yiddish. His acting career started in a Yiddish theater in Los Angeles. The well-known "live long and prosper" gesture is, in fact, a Jewish liturgical blessing and represents the letter "ש"$_{\mathrm{HE}}$, the initial of "שַׁדַּי"$_{\mathrm{HE}}$ ⌊(The) Almighty⌋. Not to mention another characteristic of the Spock character that made him utterly sympathetic: his sense of humor, as in the following exchange with Lt Uhura:

> UHURA: Tell me how planet Vulcan looks when the moon is full.
> SPOCK: Vulcan has no moon, Miss Uhura.
> UHURA: I'm not surprised, Mr. Spock. *The Man Trap* S1E1.

10.4.3.2 Predestination

The *Predestination* movie (2015) is based on Heinlein's novel *All You Zombies* (1959). It is considered the ultimate time-travel story, way beyond the Doctor's wibbly-wobbly, timey-wimey stuff. Not only do all characters move back and forth in time, but most importantly, one progressively discovers that all main characters (the baby Jane, adult Jane, John, the barman, and the Fizzle Bomber) are the same person.

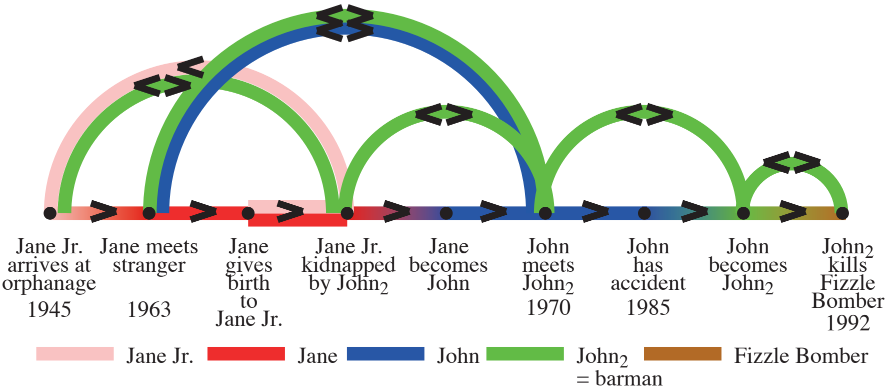


Fig. 10.2 The diagram of events and transformations of the unique character of the movie *Predestination*

The plot is as follows (see Fig. 10.2): in 1945, baby Jane arrives at the orphanage without any link to her past and origin; as a teen she shows exceptional abilities but also a tendency to be solitary; in 1963, she meets a young man that thinks and acts just like her; they have an affair but he suddenly disappears; she is pregnant by him and gives birth to a girl, which she calls Jane, like herself; the doctors announce her that she has in fact genitals of both genders but that after her birth they were forced to remove the feminine genitals and that from now on she would start a process to become a man—she chose the name John; in 1970, John enters a bar and tells his story to the barman; the barman asks him whether he would be ready to kill the man who left him/her pregnant and abandoned him/her; he agrees and they travel back to 1963 where he realizes that he is the mysterious man with whom Jane had the affair; after some time the barman reappears and takes the reluctant John back to the future; he becomes a time agent and succeeds in all his missions except one: capturing a mass murder called the Fizzle Bomber; in 1985 he is about to disarm one of the Fizzle Bomber's bombs when it explodes and burns his face; he becomes a new face, the one of the barman; it is the barman that kidnaps baby Jane and brings her to the orphanage; in 1992 he manages at last to have a confrontation with the Fizzle Bomber; he discovers that it is once again himself, setting bombs to change history and save even more lifes; he kills the Fizzle Bomber.

Not only are the five main characters of the movie the same person, but this person has sex with him/herself, gives birth to him/herself, and kills him/herself. His/her life is a closed circuit.

To represent the plot of this movie in logic, one would need special conventions, such as separating subjective time from objective time and deactivating temporal logic axioms, stating that subjective and objective times always move in the same direction. And finally, one would need to revise the irreflexivity of the parent relation, which in this case is reflexive, symmetric, and transitive.

Science Fiction is popular because it extends our ways of thinking and opens new horizons to imagination—similarly, it opens new challenges to knowledge representation systems and artificial intelligence. Representing the plot of *Predestination* in logic is not a scientific project essential to our survival. Still, it is a nice challenge that can shed light on our perception of the world and the limits of our methods.

10.5 Exercises

Exercise 9-1: Socrates’ “I know that I know nothing”

Show that Socrates’ “I know that I know nothing” “ἓν οἶδα ὅτι οὐδὲν οἶδα”$_{\text{EL}}$ is a contradiction both in FOL and in Epistemic Logic. Find an alternative that is not a contradiction.

Exercise 9-2: Use Prolog for Formal Semantics

On p. 172, we have presented the Python *nltk* implementation of formal semantics. More than thirty years before *nltk*, formal semantics have already been implemented. The programming language used was *Prolog*, which stands for “Programmation en logique”$_{\text{FR}}$ ⌊Programming in Logic⌋. Prolog has been developed by Colmerauer (Marseille) and Roussel (Sophia-Antipolis) [6], based on previous work by Kowalski (Edinburgh).

Prolog is a *declarative* language, information, and knowledge are stored in the form of relations, that can be *facts* (such as “‘Rory’ is a noun”) and *rules* (such as “a sentence consists of a noun group followed by a verb group”). Once the facts and rules are stored in what is essentially a knowledge base, one runs *queries* to trigger inferences and obtain further information. Prolog is useful in Artificial Intelligence and has long been used in NLP.[7] And Prolog is fun, with its unique way of solving problems.

Your task: use Prolog to extract the semantics of “Rory loves Amy” out of the semantics of its words, as we have already done in p. 171 (the theory) and p. 172 (in Python).

Exercise 9-3: Prove that Curiosity killed the Cat using Attempto Controlled English, *gkc*, and RACE

Russel & Norvig’s first edition [29, p. 282] (1995) included the following logical problem:

---

[7] See, for example, the excellent [24] that explores NLP using the programming languages Perl and Prolog.

1. Jack owns a dog.
2. Every dog owner is an animal lover.
3. No animal lover kills an animal.
4. Either Jack or Curiosity[8] killed the cat named Tuna.

from which one deduces that it was Curiosity who killed the cat. In subsequent editions ([30, p. 298], etc.), the problem changed slightly, but we will stick to the original version.

Consider that sentences 1–4 are a knowledge base, and let us show that when the negation of "Curiosity killed the cat" is added, we obtain a contradiction.

Translate this problem into Attempto Controlled English.

Then, follow two approaches:

1. convert from Attempto to FOL for inspection. Translate Attempto into the TPTP syntax and use the *gkc* (Graph Knowledge Core) prover [34] to show that it was indeed Curiosity who killed the cat;
2. copy-paste the Attempto text into the Attempto reasoner RACE[9] and submit the query on who killed the cat to this tool.

Exercise 9-4: The "logical proof of the existence of God" fallacy

Do you agree that the formalization of the English sentence "If God does not exist, then it is not true that if I pray, then my answers will be answered" is

$$\neg G \to \neg(P \to A), \tag{10.8}$$

(were G stands for "God exists," P stands for "I pray," and A stands for "my prayers are answered") and that this sentence and, therefore, the formula, is true?

If yes, imagine that you do not pray, i.e., the interpretation of $\neg P$ is true.

The formula (10.8) can be rewritten as $(G \vee P) \wedge (G \vee \neg A)$. For a conjunction to be true, both parts must be true. The left part is $G \vee P$, and we know that P is false since the fact that you do not pray is part of our hypothesis. Therefore, G must be true; we have just proven God exists.

What is wrong with our reasoning?

References

[1] Franz Baader and Carsten Lutz. "Description Logic". In: *Handbook of Modal Logic*. Ed. by Patrick Blackburn, Johan van Berthem, and Frank Wolter. Elsevier, 2007, pp. 757–819.

[2] Franz Baader et al., eds. *The Description Logics Handbook – Theory and Applications*. 2nd ed. Cambridge University Press, 2008.

[3] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 3rd ed. Springer, 2012.

[8] We are witnessing the personification of curiosity, similar to the personification of fortune and virtue in Monteverdi's opera "The Coronation of Poppaea."

[9] http://attempto.ifi.uzh.ch/race/

[4] Patrick Blackburn and Johan van Benthem. “Modal Logic. A Semantic Perspective”. In: *Handbook of Modal Logic*. Ed. by Patrick Blackburn, Johan van Berthem, and Frank Wolter. Elsevier, 2007, pp. 1–84.
[5] David Carson. “The Abduction of Sherlock Holmes”. In: *International Journal of Police Science & Management* 11.2 (2009), pp. 193–202.
[6] Alain Colmerauer and Philippe Roussel. “The Birth of Prolog”. In: *History of Programming Languages—II*. New York, NY, USA: Association for Computing Machinery, 1996, pp. 331–367. URL: https://doi.org/10.1145/234286.1057820.
[7] Jean-Paul Delahaye. *La logique, un aiguillon pour la pensée*. Paris: Belin. Pour la Science, 2012.
[8] Jean-Paul Delahaye. *Logique, informatique et paradoxes*. Paris: Belin. Pour la Science, 1995.
[9] Jean-Paul Delahaye. *Outils logiques pour l’intelligence artificielle*. Paris: Eyrolles, 1988.
[10] Hans-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. 3rd ed. Springer, 2021.
[11] Julia Galef. *The Scout Mindset. Why Some People See Things Clearly and Others Don’t*. Portfolio / Penguin, 2021.
[12] L.T.F. Gamut. *Logic, Language and Meaning. Volume 1. Intensional Logic and Logical Grammar*. The University of Chicago Press, 1991.
[13] L.T.F. Gamut. *Logic, Language and Meaning. Volume 1. Introduction to Logic*. The University of Chicago Press, 1991.
[14] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.
[15] Shawn Hedman. *A First Course in Logic*. Vol. 1. Oxford Texts in Logic. Oxford University Press, 2004.
[16] C. Maria Keet. *An Introduction to Ontology Engineering*. Vol. 20. Computing. College Publications, 2018.
[17] C. Maria Keet. “The African wildlife ontology tutorial ontologies”. In: *Journal of Biomedical Semantics* 11 (2020), p. 4.
[18] Robert Kowalski. *Computation Logic and Human Thinking*. Cambridge University Press, 2011.
[19] René Lalement. *Computation as Logic*. Translated by John Plaice. Prentice Hall, 1993.
[20] Erik T. Mueller. *Commonsense Reasoning*. San Francisco: Morgan Kaufmann, 2006.
[21] Erik T. Mueller. “Event Calculus”. In: *Handbook of Knowledge Representation*. Ed. by F. van Harmelen, V. Lifschitz, and B. Porter. Elsevier, 2008, pp. 671–708.
[22] Leonard Nimoy. *I am Spock*. Hyperion, 1995.
[23] Catarina Dutilh Novaes. *Formal Languages in Logic. A Philosophical and Cognitive Analysis*. Cambridge University Press, 2012.
[24] Pierre M. Nugues. *Language Processing with Perl and Prolog. Theories, Implementation, and Application*. 2nd ed. Cognitive Technologies. Springer, 2014.
[25] Marek Perkowski. “Modal Logic and its Applications, Explained using Puzzles and Examples”. In: *Systems Science Friday Noon Seminar Series* 60 (2011).
[26] Stefano Piraino et al. “Reversing the life cycle: medusae transforming into polyps and cell transdifferentiation in Turritopsis nutricula (Cnidaria, Hydrozoa)”. In: *Biological Bulletin* 190 (1996), pp. 302–312.
[27] Williard Van Orman Quine. “Quantification and the Empty Domain”. In: *The Journal of Symbolic Logic* 19 (1954), pp. 177–179.
[28] Jiří Raclavský. “The Barber Paradox: On its Paradoxicality and its Relationship to Russell’s Paradox”. In: *Prolegomena* 13.2 (2014), pp. 269–278.
[29] Stuart Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. 1st ed. Prentice Hall, 1995.
[30] Stuart Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. 2nd ed. Prentice Hall, 2002.

[31] Klaus Schild. “Combining terminological logics with tense logic”. In: *Proc. of the 6th Portuguese Conf. on Artificial Intelligence (EPIA'93)*. Vol. 727. Lecture Notes in Computer Science. Springer, 1993, pp. 105–120.

[32] John Stuart Mill. *A system of logic, ratiocinative and inductive: being a connected view of the principles of evidence, and methods of scientific investigation*. 1st ed. Vol. I. London: John W. Parker, 1843.

[33] Gaisi Takeuti and Wilson M. Zaring. *Introduction to Axiomatic Set Theory*. Vol. 1. Graduate Texts in Mathematics. Springer, 1971.

[34] Tanel Tammet. “GKC: A Reasoning System for Large Knowledge Bases”. In: *Automated Deduction – CADE 27*. Ed. by Pascal Fontaine. Springer, 2019, pp. 538–549.

[35] David A. Wheeler. “The Origin of All Men are Mortal”. https://dwheeler.com/essays/all-men-are-mortal.html. 2023.

Chapter 11
Ontologies and Conceptual Graphs

In the previous chapter, we have presented Description Logics. In this chapter, we turn to *ontologies*, i.e., " 'implementation versions' of description logics with rich tooling support" [17, p. 67]. The specific kind of ontologies we are going to present are ontologies OWL (Web Ontology Language) format, and the "tooling support" is an extensive framework called the *Semantic Web*.

11.1 The Semantic Web

> [the Semantic Web] remains a mystery to a large percentage of Internet users.
>
> Dantuluri *et al.* [11, p. 3867]

11.1.1 The Origins

People usually connect Tim Berners-Lee's invention of the Web with his internal CERN memo released in March 1989 [3]. However, the process started quite earlier. In 1966, Charles F. Goldfarb was a Harvard Law School attorney practicing in Boston, MA. His hobby was "rallymastering," i.e., writing route instructions for sports car rallies [14]. He did it in such a well-organized way that eventually, some friend told him that his rally instructions looked like computer programs. That was enough to make him join IBM, and in 1969, he invented GML (Generalized Markup Language). GML later became SGML (S for Standard), also known as ISO 8879, and still later, XML (X for eXtensible). Therefore, SGML and Tim Berners Lee's ☞ 359
HTML are profoundly rooted in the Chomskyan tradition. In SGML and HTML, a document is considered a word in a formal language, the (formal) grammar of which is contained in a part of the SGML document called DTD ("Document-Type Definition"). An SGML document is a set of "elements" with a tree structure based on the nesting relation. At the leaf level of the tree, we have either empty elements

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_11

(potentially with attributes) or text fragments, processing instructions, raw (binary) data or comments.

SGML was a lot more general than XML. At the beginning of an SGML document, one could redefine all special markup characters. People allergic to tags looking like `<body>` could choose to represent them by `$body£` if they wished. Even worse, both opening and closing tags could be optional. For example, if tag `<b>` would always be introduced between `<a>` and `<c>`, then one could define it as *optionally* represented. It may seem unbelievable, but this would mean that the tag `<b>` is there without being there. Besides being an author's nightmare since one would need to know the DTD well enough to know where optional tags are hidden, this also means that a document without a DTD was useless since important information was hidden there. The differences between SGML and XML are a fascinating subject. The reader can find more information in Bradley [8, Chap. 32].

An HTML document, the basic building block of the first version of the WWW, is an SGML document with a fixed DTD containing 18 element types (in its first version, for all versions of HTML, see the WHATWG Web site[1]). The semantics of these elements represent standard document structure (sections of various levels, paragraphs, tables, lists, images), typographical styles (italics, bold, underlined, etc.), hypertext links, etc. To be called an "HTML document," a document had to be *well-formed*, respecting SGML syntax, and *valid*, being a word in the formal grammar described by HTML's DTD. Usual actions on HTML documents were checking their validity, displaying them in a Web browser's window, and navigating through them.

11.1.2 The Semantic Web Cake

In 2000, Tim Berners Lee introduced, in a presentation's slide,[2] a roadmap for the future of the Web. After six years of gestation, he released an official description of this new kind of Web, the *Semantic Web*, in a paper that was the first paper of the first volume of a new journal called *Foundations and Trends in Web Science* (the term "Web science" did not catch on, but the journal is still active today). In this 130-page-long manifesto, he presented a revised roadmap called the "Semantic Web Cake." We display it in Fig. 11.1.

For our purposes, we can divide the Semantic Web Cake into three parts: the lower layers (Unicode, URI, XML), which we present in Chapter 13; the middle layers (RDF, RDF-S, OWL, SPARQL, RIF), which will be the primary concern of this chapter; and the higher layers, as well as the vertical one, which does not yet exist.

[1] `https://html.spec.whatwg.org/`

[2] `http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`

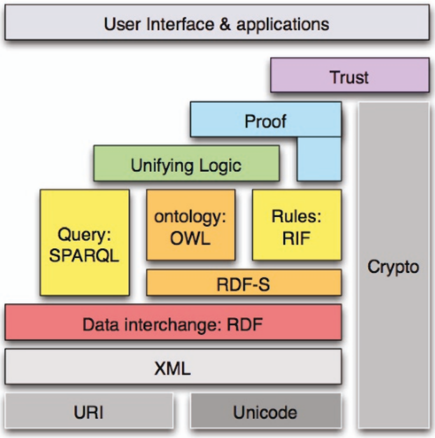


Fig. 11.1 The revised Semantic Web Cake, as represented in Berners Lee *et al.* [5, p. 22]

11.1.3 The Lower Levels: Unicode Literals, URIs, XML

Unicode is the currently prevalent character encoding. As such, it maps "characters" to numerical values (called "code points"). A "character" functionally corresponds to a grapheme [15, pp. 139–142] but is defined in Unicode in a completely different way. We describe Unicode in Chapter 12 as a data format. ☞ 343

In the Semantic Web Cake, Berners Lee legitimates Unicode as the only valid encoding and considers it the most basic layer of the cake.

Interestingly, at the same level as Unicode characters, we have URIs. According to [4],

> A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. [4]

Before URIs there were URLs (Uniform Resource Locators). The difference is as follows, and it is important:

> Nor should it be assumed that a system using URIs will access the resource identified: in many cases, URIs are used to denote resources without any intention that they be accessed. [4]

In other words, a URI is an identifier of something that may or may not point to a resource. If it does not point to a resource, then it is just an identifier like there are Social Security numbers for humans. The difference with a Social Security number is that a single authority issues the latter, the Social Security Administration, which can take measures to ensure uniqueness, while anyone can write the former. The trick with URIs is that uniqueness is based on an Internet domain name and extension. An organization or individual with a domain name can use it to write an arbitrary number of URIs using the syntax of Unix paths following the domain name and extension. In other words, a URI is formally indistinguishable from a URL, but it serves for *identifying* resources and not for *accessing* resources. The format of a URI [4, §3] is of the form

```
scheme : hierarchical part [? query] [# fragment]
```

In the case of an URL, the hierarchical part is the machine name, domain name, domain extension, and Unix path to the resource, the query part is used for queries to CGI scripts, and the fragment part is a label that points to a part of the HTML page.

URIs are written in ASCII, but there are also IRIs, namely Internationalized Resource Identifiers [12], which are Unicode-based URIs. And IRI may use Unicode characters, but applications such as Web browsers are requested to convert IRIs into URIs in the following way: the bytes of a UTF-8 encoded representation of the characters are in hexadecimal form, so for example, the IRI

```
http://www.πολυτονικό.org
```

becomes

```
http://www.%CF%80%CE%BF%CE%BB%CF%85%CF%84%CE%BF%CE%BD%CE%B9%CE%BA%CF%8C.org
```

where each Greek Unicode character is represented by two bytes in UTF-8 and, therefore, by two hexadecimal numbers. Conversion from IRI to URL is a good way to fight against phishing, namely the use of homographic Unicode characters. For example, a Cyrillic “o”$_{\text{RU}}$ is visually identical to a Latin “o,” and an IRI `http://www.amazon.com` with a Cyrillic “o”$_{\text{RU}}$ could otherwise convince the user that it is the address of the Web site of a well-known company and ask for her credit card credentials… Deception becomes more difficult by expanding the IRI into the URL `http://www.amaz%D0%BEn.com`.

☞ 359 Above URIs and Unicode, there is XML. An XML document is a (formal) word in a formal language described by a DTD or an XML Schema. This formal language's alphabet comprises elements, attributes, text contents, processing instructions, CDATA sections, and comments. We describe XML in Chapter 13.1 as a data format.

An important aspect of XML is the notion of *namespace*. There is a widespread belief that XML is only about syntax. This is not the case with namespaces: a namespace is a unique identifier that refers to some standard framework. For example, when we need to specify a data type unambiguously, and this data type is part of the XML Schema Data Types framework, we refer to the namespace `http://www.w3.org/2001/XMLSchema-datatypes` through a prefix (for example `xs`). To define what a Person is, their name, surname, family name, given name, etc., one can use the namespace of the FoaF (Friend of a Friend) ontology, which is `http://xmlns.com/foaf/0.1/`. Defining, even directly, the meaning of XML elements or attributes through namespaces is a semantic annotation task, be it rudimentary.

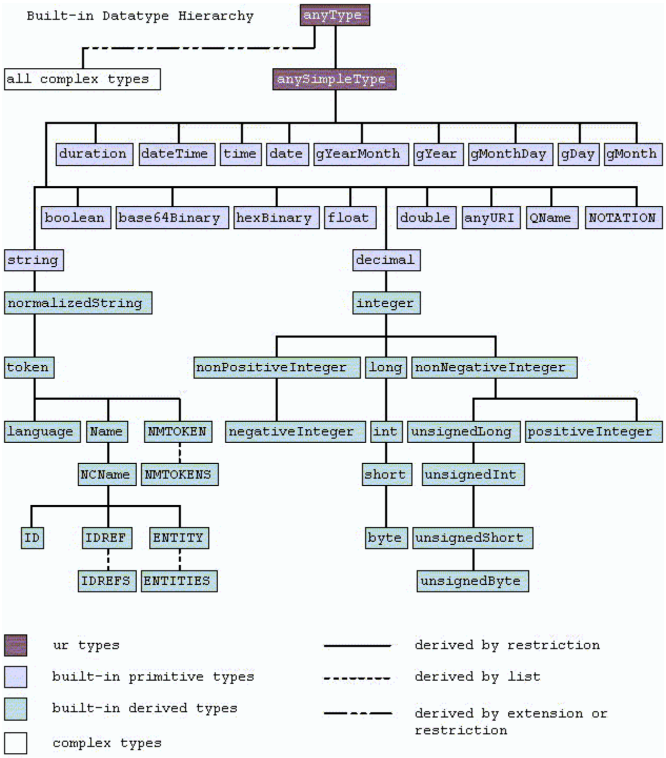


Fig. 11.2 The hierarchy of XML Schema datatypes, from [6]. The namespace to access these types is `http://www.w3.org/2001/XMLSchema-datatypes`

11.1.4 RDF, SPARQL, and RDFS

The entire Semantic Cake is based on the notion of *RDF triple*, where RDF stands for Resource Description Framework. As its name suggests, an RDF triple has three components:

1. a *subject*, which is either an IRI or a *blank node*;
2. a *predicate*, which is an IRI;
3. an *object*, which is either an IRI, or a *literal*.

For example, imagine the author wants to express that he lives in Brest (France). It happens that the author is represented by a node in Wikidata (an online depository containing 107.2M RDF triples as of October 2023), namely Q102323864. He, therefore, can use an IRI involving the Wikidata namespace:

`http://www.wikidata.org/entity/Q102323864`

By searching in the list of 11,253 (as of October 2023) predicates (called "properties") used in Wikidata, he found that the one that comes closest to the act of living in some location is P551, called “residence” and described as “the place where the person is or has been, resident.” The corresponding IRI is

```
http://www.wikidata.org/prop/direct/P551
```

And finally, there is a Wikidata IRI for Brest (that allows us to distinguish among municipalities in France, Belarus, Germany, Bulgaria, FYROM, Serbia, Slovenia, and Croatia), namely

```
http://www.wikidata.org/entity/Q12193
```

In the following, we will use a human-friendly notation for RDF triples and queries, called *Turtle* (Terse RDF Triple Language) [2].

As IRIs tend to be verbose, we use prefixes to abbreviate them:

```
@prefix wd: http://www.wikidata.org/entity/ .
@prefix wdt: http://www.wikidata.org/prop/direct/ .
```

The RDF triple that states that the given person resides in the given municipality is then

```
wd:Q102323864 wdt:P551 wd:Q12193 .
```

☞ 347 A *literal*, in the simplest case, is a string of Unicode characters (in NFC) delimited by double quotes. In the case where a specific data type is needed, one can add a datatype IRI (two hats followed by the IRI of the data type), as in the example

```
"1"^^xs:nonNegativeInteger
```

where we use the prefix `xs` for the XML Schema Datatypes namespace (see Fig. 11.2).
☞ 349 It is also possible to use IETF BCP47 language tags for literals, such as

```
"Gift"@de-DE
```

to assign language German (spoken in Germany) to the word “Gift”$_{\text{DE}}$ (which therefore means “poison” and not “item offered on birthdays”).

Now that we know what RDF triples look like, we may want to search in an RDF depository. For that, there is a query language called *SPARQL* (a recursive initialism: SPARQL Protocol and RDF Query Language) [1]. A query in SPARQL consists of three parts:

1. the prefixes used in the RDF triple patterns of the query,
2. the variables used in the query,
3. the RDF triple patterns that have to be satisfied.

The SPARQL engine will find all triples that satisfy the query triple patterns and will return the values of the variables used in the query for each set of triples found.

Here is an example: let us request the population of the city of Brest (France):

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
SELECT ?population
WHERE
{
  wd:Q12193 wdt:P1082 ?population .
}
```

By entering this query in the Wikidata SPARQL portal[3] we get the result

```
population
139926
```

Now let us request the populations (and countries) of all municipalities, the names of which start with "Brest." For that, we will use predicate P1448 for the town's name, predicate P17 for the country in which the town is located, predicate P1705 for the country's name, and a regular expression for keeping only town names that start with `Brest` (Latin alphabet) or `Брест` (Cyrillic alphabet):

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
SELECT ?population ?townname ?countryname
WHERE
{
  ?town wdt:P1448 ?townname .
  ?town wdt:P1082 ?population .
  ?town wdt:P17 ?country .
  ?country wdt:P1705 ?countryname .
  FILTER REGEX (?townname, "^(Brest|Брест)")
}
```

and the result (after having removed some lines) will be:

```
population    townname          countryname
355           Брестів           Україна
448           Brestov           Slovenská republika
627           Brestot           République française
1001          Brest             Österreich-Ungarn
2658          Brestovany        Slovenská republika
3726          Brestovac         Republika Hrvatska
139926        Brest             République française
213880        Brest Métropole   République française
342461        Брест             Рэспубліка Беларусь
378492        Brest             République française
```

Besides the predicates we can take from ontologies such as Wikidata, there are also hardcoded predicates in RDF and an RDF extension called RDFS (RDF Schema) [9]. These allow operations such as defining classes and roles (called *properties* in this context), asserting that some class is a subclass of another class, that some individual belongs to a class, that the domain and range of a given property belong to given classes, etc.

For example, to assert that, in some ontology called `http://example.org/#`, there is (1) `Ashildr` who is a person (the class taken from FoaF), but also a member of the class of lovely persons and (2) a property called `love`, the domain of which are persons and the range of which are lovely persons, we will write

```
@prefix me: <http://www.example.org/#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

[3] https://query.wikidata.org/

```
me:LovelyPerson rdf:type rdfs:Class .
me:LovelyPerson rdfs:subClassOf foaf:Person .
me:Alice rdf:type me:LovelyPerson .
me:loves rdfs:domain foaf:Person .
me:loves rdfs:range me:LovelyPerson .
```

An important property of RDF is *reification*, from the Latin “res”$_{\text{LA}}$ ⌊thing⌋, “turning something into a thing.” What can be converted into a “thing” is an RDF triple. It suffices to create it as an instance of the class `Statement` and to define its subject, predicate, and object. Once a statement has been created, it can be used in other RDF triples. For example, to say that the statement “Groucho Marx wrote *Das Kapital*” is a lie, and knowing that the Wikidata entry of Groucho Marx is Q103846, the one of *Das Kapital* is Q58784, the one of lies is Q4925193, the one of the predicate “is an instance of” is P31, and that, in another ontology called *Dublin Core*, there is a predicate of creator, one can write:

```
@prefix dc: <"http://purl.org/dc/elements/1.1/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix wd: <http://www.wikidata.org/entity/>
@prefix wdt: <http://www.wikidata.org/prop/direct/>

_:x rdf:type rdf:Statement .
_:x rdf:subject wd:Q103846 .
_:x rdf:predicate dc:creator .
_:x rdf:object wd:Q58784 .
_:x wdt:P31 wd:Q4925193 .
```

where `_:x` is a “blank node identifier,” in the sense that we use a common identifier in these triples but will neither give it a unique name nor use it elsewhere. Reification is important because in the previous code, neither Groucho Marx nor *Das Kapital* are lies, but the statement that the former authored the latter. By writing the statement directly, as here:

```
@prefix dc: <"http://purl.org/dc/elements/1.1/>
@prefix wd: <http://www.wikidata.org/entity/>

wd:Q103846 dc:creator wd:Q58784 .
```

we would be unable to assert that the statement *per se* is a lie.

11.1.5 OWL Ontologies

The word “ontology” means “study of the on,” where “ὄν”$_{\text{EL}}$ in Greek is the neuter form of the present participle of the verb “to be,” i.e., “being.” It is, therefore, the “study of what is,” and one can hardly consider something more general than that. In philosophy, ontology is defined as

> the study of reality as constructed in both human and non-human worlds (Wikipedia)

where the “non-human” adjective presumably refers not to aliens but to pets. Important in this definition is the word “constructed.” This acknowledges that in the flow of perceptions we are experiencing, reality is not a hard, immutable fact but a construction of every moment. This may be the common point of “ontology” in philosophy and “ontologies” in computer science. In both cases, there is a construction of some reality. In the former case, it is by living in this reality. In the latter case, it is by describing reality so that artificial intelligence can access it.

In computer science, ontologies mainly describe states of the world (the real world or some imaginary world) and are important tools of knowledge representation.

For Prévot *et al.*, [19, p. 3], ontologies are defined as

> specifications of shared conceptualizations [of a given domain],

which means that we have two steps:

> conceptualization is the relevant informal knowledge one can extract and generalize from experience, observation, or introspection [and] specification is the encoding of this knowledge in a representation language. [19, p. 3]

In our case, the representation language is OWL (Web Ontology Language), and more specifically, OWL 2 (in the following, we will omit the version, but we will always mean version 2 of OWL), which is quite a particular language since it has a large number of equivalent syntaxes for the same semantics. It is essentially a typed description logic, defined so that every formula can be translated into RDF triples. There are two OWL variants: OWL-DL is more description logic-oriented, and OWL Full is more graph-oriented, but the differences are very slight [16, § 9], and we will not consider them here.

As we have already described description logics in the previous chapter, what follows will be mainly about OWL’s syntax to answer the question: how are the various ingredients of description logics implemented in OWL?

We already mentioned several ways of representing OWL ontologies: initially the Web consortium privileged an XML representation of OWL, but later, it moved to a format based on Turtle. Besides that, the OWL specifications, the OWL user’s manual, and the OWL primer use yet another syntax called “Functional-Style Syntax.” This syntax has been implemented by *Protégé*, the most popular tool for editing and creating OWL ontologies, so that it has, maybe inadvertently, become yet another regular syntax for OWL documents.

To compare the Functional-Style (from now on, FS) and the Turtle notation, let us take the description logic formula

$$\mathsf{Leaves} \sqsubset \exists \mathsf{isPartOf}.\mathsf{Branches},$$

saying that “leaves are parts of branches,” and more precisely, that for each individual in the class of leaves, there is at least one individual in the class of branches. In (Protégé-generated) FS, this is represented as follows:

```
SubClassOf(:Leaves ObjectSomeValuesFrom(:isPartOf :Branches))
```

Here the subsumption $\sqsubseteq$ is a first binary relation between the class `:Leaves` (the `:` denoting the empty prefix, which is representing the current ontology) and a class obtained by a function `ObjectSomeValuesFrom` (i.e., $\exists$), applied to the role `:isPartOf` and the class `:Branches`. As we see, apart from the fact that is written as a predicate with two arguments, this notation is quite close to the formula.

The Turtle representation of the same formula is the following:

```
:Leaves rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :isPartOf ;
                          owl:someValuesFrom :Branches
                        ] .
```

Here, the same object `:Leaves` participates to a first triple, asserting that it is of type `owl:Class`, and then to a second triple, asserting that it is a subclass of a constructed class. This constructed class is obtained by an *abbreviated construction*, that works as follows: every time a bracket is opened, a blank node is created, and this blank node is used as a subject in a new triple together with the two other parts of the bracketed expression. In other words, if we write

```
[ loves Rory ] name Amy
```

then we create a first triple `_ name Amy` (a blank node replaces the bracketed expression) and a second triple `_ loves Rory` (the blank node becomes the subject of a triple made by what is contained in the bracketed expression).

To return to the example of leaves and branches, when we expand these abbreviations, we get the following five triples:

```
:Leaves rdf:type owl:Class .
:Leaves rdfs:subClassOf _ .
_ rdf:type owl:Restriction .
_ owl:onProperty :isPartOf .
_ owl:someValuesFrom :Branches .
```

where the existential restriction is described by three triples and is used as an object in a subsumption triple, the subject of which is `:Leaves`, which was previously defined as being a class.

The semantics of these two code excerpts are the same, but the former is rather description logic-oriented while the latter is RDF triple-oriented.

Let us now stick to FS notation and see how the various ingredients of description logic are represented. As *Protégé* has become an indispensable tool, we will also show Protégé screen captures for our example of AWO.

11.1.5.1 Prefixes, Ontology

In FS, prefixes are included at the beginning of the document in the form of unary predicates, the arguments of which are prefixes followed by a colon and a namespace:

```
Prefix(:=<http://www.fluxus-virus.com/ontologies/2023/4/AWO#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
```

After the list of prefixes, a function `Ontology` is invoked, the first entry of which is the namespace of the ontology itself:

```
Ontology(<http://www.fluxus-virus.com/ontologies/2023/4/AWO#>

...

)
```

Included in the argument of `Ontology` are all declarations and formulas of the ontology.

11.1.5.2 Classes, Roles, Individuals

The first part of the ontology is a list of declarations of classes, roles, and individuals:

```
Declaration(Class(:Animals))
Declaration(Class(:Branches))
Declaration(Class(:Carnivores))
Declaration(Class(:Giraffes))
Declaration(Class(:Herbivores))
Declaration(Class(:Leaves))
Declaration(Class(:Lions))
Declaration(Class(:Plants))
Declaration(Class(:Trees))
Declaration(ObjectProperty(:eats))
Declaration(ObjectProperty(:isPartOf))
Declaration(NamedIndividual(:Gigi))
Declaration(NamedIndividual(:Léon))
```

where we have classes for animals, branches, etc., in alphabetical order, *object properties* (aka roles) for eating and being part of, and individuals `Gigi` and `Léon`. No information is given about these elements. They are just declared.

We have called this in description logics, the ABox of the knowledge base.

11.1.5.3 Subsumptions, Restrictions

Follows a section where the various classes are described. Here are the correspondences between description logic symbols and FS functions:

$\top$	`owl:Thing`
$\bot$	`owl:Nothing`
$C_1 \sqsubset C_2$	`SubClassOf(C1 C2)`

$R_1 \sqsubset R_2$	`SubObjectPropertyOf(R1 R2)`
$C_1 \equiv C_2$	`EquivalentClasses(C1 C2)`
$\neg C$	`ObjectComplementOf(C)`
$C(I)$	`ClassAssertion(C I)`
$R(I_1, I_2)$	`ObjectPropertyAssertion(R I1 I2)`
$\neg R(I_1, I_2)$	`NegativeObjectPropertyAssertion(R I1 I2)`
$C_1 \sqcap C_2$	`ObjectIntersectionOf(C1 C2)`
$C_1 \sqcup C_2$	`ObjectUnionOf(C1 C2)`
R^{-1}	`ObjectInverseOf(R)`
$\exists R.C$	`ObjectSomeValuesFrom(R C)`
$nR.C$	`ObjectExactCardinality(n R C)`
$\geq nR.C$	`ObjectMinCardinality(n R C)`
$\leq nR.C$	`ObjectMaxCardinality(n R C)`
$\exists R.\mathsf{Self}$	`ObjectHasSelf(R)`
$\forall R.C$	`ObjectAllValuesFrom(R, C)`

11.1.5.4 Domain, Range, Identity

We can also assert constraints on the domain and range of every role. Furthermore, we can say that two classes are disjoint, that two names refer to different individuals or that they refer to the same individual.

The domain of R is C	`ObjectPropetyDomain(R C)`
The range of R is C	`ObjectPropetyRange(R C)`
Classes C_1 and C_2 are disjoint	`DisjointClasses(C1 C2)`
Individuals I_1 and I_2 are the same	`SameIndividual(I1 I2)`
Individuals I_1 and I_2 are different	`DifferentIndividuals(I1 I2)`

11.1.5.5 Role properties

In DL, it was not very clear how to do it. In OWL, we can request that a role be reflexive, irreflexive, symmetric, asymmetric, and/or transitive:

R is reflexive	`ReflexiveObjectProperty(R)`
R is irreflexive	`IrreflexiveObjectProperty(R)`
R is symmetric	`SymmetricObjectProperty(R)`
R is asymmetric	`AsymmetricObjectProperty(R)`
R is transitive	`TransitiveObjectProperty(R)`

Transitive closure is not available in OWL since its description logic is $\mathcal{SHOIN}$ (no symbol $\mathcal{R}^+$), but see Bonatti [7] for a possible extension of OWL.

11.1.5.6 Datatypes

OWL's main extension to description logic is that it is typed, and we can assign attributes to individuals, the values of which will be of specific types. For example, to assert the age of a person, we can use a special kind of role called "data property" (as opposed to "object property") and request that the range of this role is the set of integers:

J is 51	`DataPropertyAssertion(:hasAge :J "51"^^xsd:integer)`
J is not 53	`NegativeDataPropertyAssertion(:hasAge :J "53"^^xsd:integer)`
Age applies to persons	`DataPropertyDomain(:hasAge :Person)`
Age $\in \mathbb{N}$	`DataPropertyRange(:hasAge xsd:nonNegativeInteger)`

11.1.5.7 Example

It is now fairly easy to understand the code of the formulas defining herbivores:

```
EquivalentClasses(:Herbivores
   ObjectIntersectionOf(
      ObjectSomeValuesFrom(:eats owl:Thing)
      ObjectAllValuesFrom(:eats
         ObjectUnionOf(:Plants ObjectSomeValuesFrom(:isPartOf :Plants))
         )
      )
   )
SubClassOf(:Herbivores :Animals)
```

11.1.6 *Protégé*

Protégé [13] is an open-source ontology editor developed at Stanford University and first released in 1999. It is available on three platforms as well as online.

We display in Fig. 11.3, in the `Classes` tab, the tree structure of the classes of our AWO as well as the two formulas defining herbivores; in Fig. 11.4, in the `Object properties` tab, the reader can see the tree structure of roles as well as the three properties of the `isPartOf` role (namely transitivity, asymmetry, and irreflexivity); in Fig. 11.5 a graphical representation of our AWO ontology produced by the *OntoGraf* plug-in.

11.1.7 Using Description Logic Reasoners with Python

Let us use two very popular reasoners, namely FACT++[4] and HermiT[5] with Python.

[4] `http://owl.cs.manchester.ac.uk/tools/fact/`

[5] `http://www.hermit-reasoner.com/`

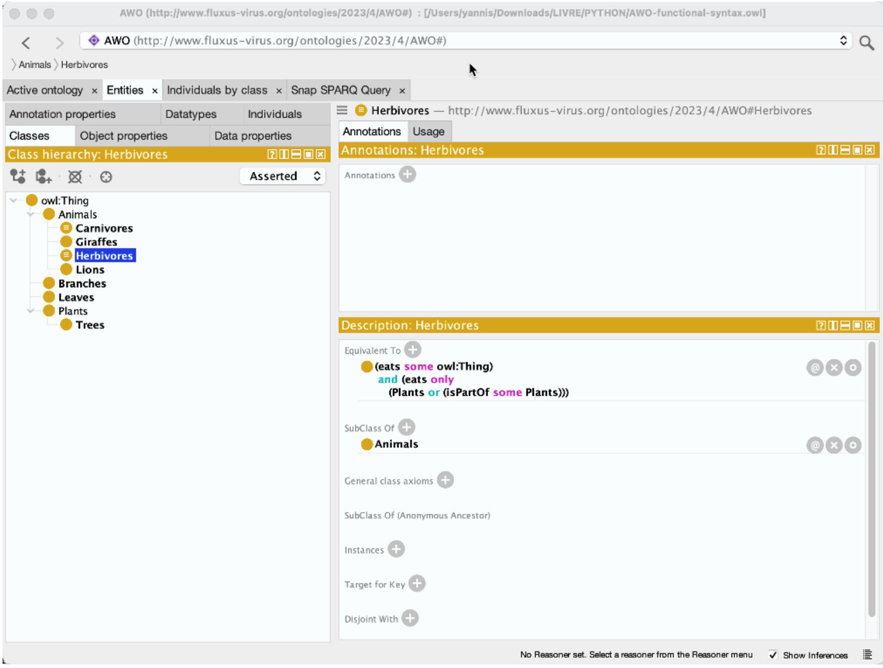


Fig. 11.3 The Classes tab of *Protégé*. In the lower-right panel, the representation of the formulas $\texttt{Herbivores} \equiv \exists\texttt{eats}.\top \sqcap \forall\texttt{eats}.(\texttt{Plants} \sqcup \exists\texttt{isPartOf}.\texttt{Plants})$ and $\texttt{Herbivores} \sqsubseteq \texttt{Animals}$

11.1.7.1 All Men Are Mortal, in FACT++

Let us create a small ontology with classes `Human` and `Mortal`, where we have $\texttt{Human} \sqsubseteq \texttt{Mortal}$, and an individual `Socrates` belonging to class `Human`. This ontology will look like this in (*Protégé*-generated) FS:

```
Prefix(:=<http://www.fluxus-virus.com/socrate/>)

Ontology(<http://www.fluxus-virus.com/socrate>

Declaration(Class(:Human))
Declaration(Class(:Mortal))
Declaration(NamedIndividual(:Socrates))

SubClassOf(:Human :Mortal)

ClassAssertion(:Human :Socrates)
)
```

To use this ontology with FACT++, we have to convert it into RDF/XML format, and this can be done easily by simply opening it in *Protégé* and saving it with the new format. Let us call the resulting file `socrates.owl`. We will use a Python package dedicated to FACT++, called *pyfactxx*. With the following code:

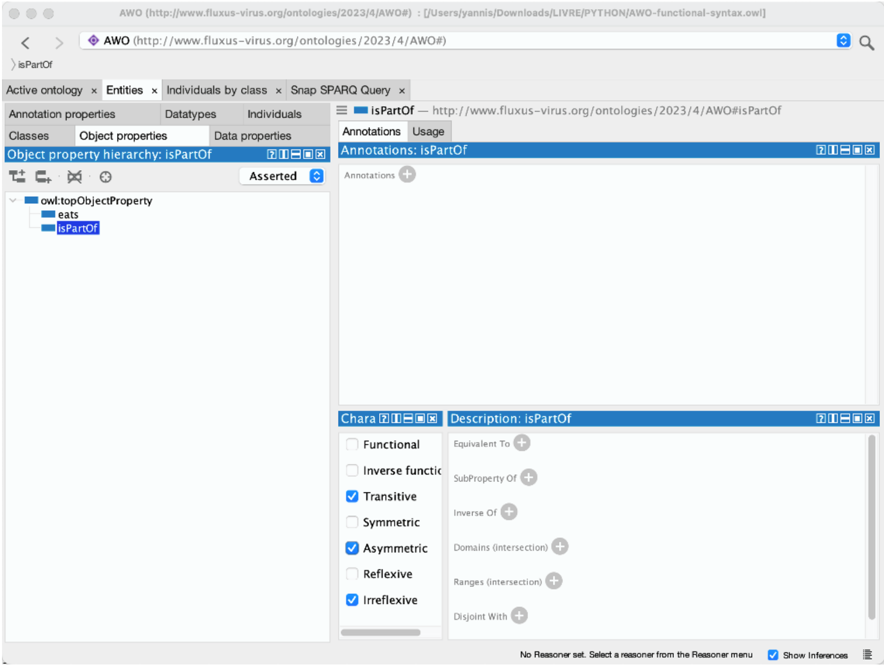


Fig. 11.4 The Object properties tab of *Protégé*. On the right, the properties of transitivity, asymmetry, and irreflexivity of the `isPartOf` role

```
from pyfactxx import coras

filename = "socrate.owl"
query = 'select ?p where { ?p rdf:type \
    <http://www.fluxus-virus.com/socrate#Mortal> . }'

crs = coras.Coras()
crs.load(filename, format="xml")
crs.parse()
crs.realise()

print(*list(crs.query(query)), sep='\n')
```

we address a SPARQL query to the reasoner, asking it to return all individuals belonging to class `Mortal`. The answer is:

```
FaCT++.Kernel: Reasoner for the SROIQ(D) Description Logic, 64-bit
Copyright (C) Dmitry Tsarkov, Ivan Rygaev, 2002-2022. Version 1.8.1
(rdflib.term.URIRef('http://www.fluxus-virus.com/socrate#Socrates'),)
```

confirming, once again, our conviction that Socrates is mortal.

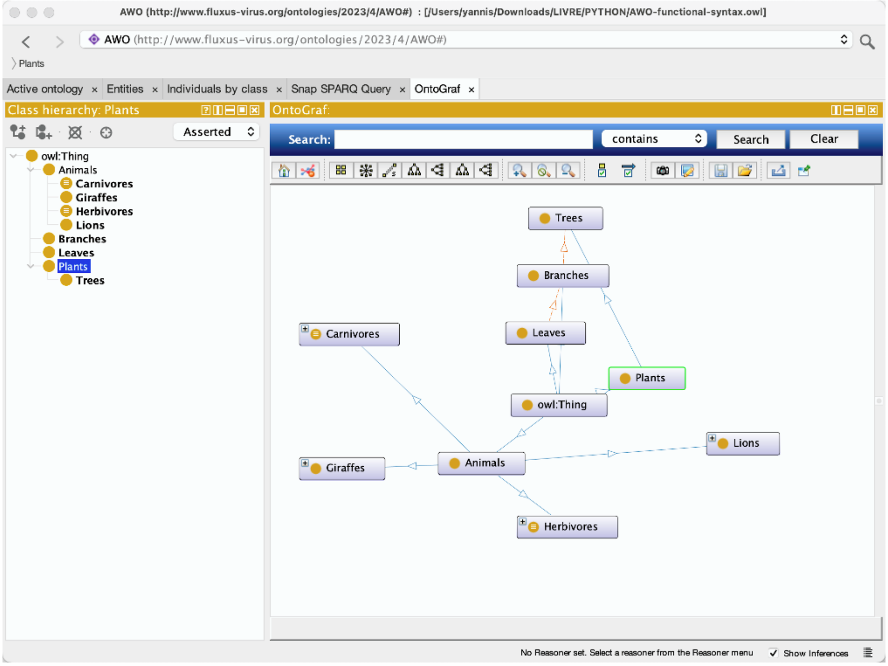


Fig. 11.5 The OntoGraf tab of *Protégé*. A graphical representation of the AWO ontology produced by the OntoGraf plug-in

11.1.7.2 Pizzas and Pepper Allergy, in HermiT

Imagine a pizza recommendation system that answers questions about pizzas in orderly (i.e., controlled) natural language. Imagine the following situation:

- we have stored the ingredients of some pizza types in our ontology: the Con Algio pizza contains garlic and rosemary, the Piccante pizza contains rosemary and salami,
- we have stored properties of each ingredient: salami is meat, garlic, and rosemary are vegetables,
- furthermore, salami contains pepper,
- and we have a general rule about a pizza not being vegetarian: *when at least one ingredient belongs to the meat class, the pizza is not vegetarian*;
- finally, we use an important property of the `contains` relation: it is transitive[6].

Here is how it looks in (*Protégé*-generated) FS:

```
Prefix(:=<http://www.fluxus-virus.com/ontologies/pizza/>)
```

[6] This is a fact that is valid in logic, but a bit counter-intuitive in natural language: a sentence such as "a building contains red blood cells" will sound weird, even though it is logically sound; a building can contain people, which contain blood, which contains red blood cells… See also Chomsky's finger on p. 138.

```
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)

Ontology(<http://www.fluxus-virus.com/ontologies/pizza>

Declaration(Class(:Meat))
Declaration(Class(:Nonvegetarian))
Declaration(Class(:Pizza))
Declaration(Class(:Vegetable))
Declaration(ObjectProperty(:contains))
Declaration(NamedIndividual(:ConAglio))
Declaration(NamedIndividual(:Garlic))
Declaration(NamedIndividual(:Pepper))
Declaration(NamedIndividual(:Piccante))
Declaration(NamedIndividual(:Rosemary))
Declaration(NamedIndividual(:Salami))

TransitiveObjectProperty(:contains)

DisjointClasses(:Meat :Vegetable)

EquivalentClasses(:Nonvegetarian ObjectSomeValuesFrom(:contains :Meat))

ClassAssertion(:Pizza :ConAglio)
ObjectPropertyAssertion(:contains :ConAglio :Garlic)
ObjectPropertyAssertion(:contains :ConAglio :Rosemary)

ClassAssertion(:Vegetable :Garlic)

ClassAssertion(:Vegetable :Pepper)

ClassAssertion(:Pizza :Piccante)
ObjectPropertyAssertion(:contains :Piccante :Rosemary)
ObjectPropertyAssertion(:contains :Piccante :Salami)

ClassAssertion(:Vegetable :Rosemary)

ClassAssertion(:Meat :Salami)
ObjectPropertyAssertion(:contains :Salami :Pepper)

)
```

A client comes along and asks, “Is there a non-vegetarian pizza with rosemary?”

Let us use a different Python package, namely *owlready2*, which can load both the HermiT and Pellet[7] reasoners and provides many methods for exploring or creating ontologies. This package is very thoroughly described in Lamy [18].

We will save our ontology in RDF/XML format. Let us call it `pizza.owl`. Here is the code to address a SPARQL query to the reasoner:

[7] https://github.com/stardog-union/pellet

```
1 from owlready2 import *
2 import rdflib.plugins.sparql as sparql
3 onto = get_ontology("pizza.owl").load()
4 sync_reasoner(infer_property_values = True)
5 graph = default_world.as_rdflib_graph()
6 query = sparql.prepareQuery("""
7            PREFIX : <http://www.fluxus-virus.com/ontologies/pizza#>
8            SELECT ?p
9            { ?p rdf:type :Nonvegetarian .
10           ?p :contains :Rosemary . }
11     """)
12 qres = graph.query(query)
13 for q in qres:
14     print(str(q[0]))
```

Some explanations: We use two packages, *owlready2* and *rdflib*. The latter provides us with SPARQL access. On line 3, we load the ontology from disk; on line 4, we ask the reasoner to perform inferences. What may be surprising is line 5: where does `default_world` come from?

We can load the same ontology several times in different "worlds." We could define a new world as

```
Gallifrey = World()
```

then load the ontology again as

```
onto2 = Gallifrey.get_ontology("pizza.owl").load()
```

and use `onto` and `onto2` as two different avatars of the same ontology, modifying them differently. By not specifying the world, we get the "default world." Line 5 moves our data from the world of OWL ontologies into the world of RDF graphs, and it is to a graph that the SPARQL query, which we prepare on lines 6–11, is applied. Finally, `str` is a list containing as many items as there are variables in the `SELECT`.

The output of this code is

```
* Owlready2 * Running HermiT...
* Owlready2 * HermiT took 1.0994329452514648 seconds
http://www.fluxus-virus.com/ontologies/pizza#Piccante
```

asserting that `Piccante` is a non-vegetarian pizza containing rosemary.

This example shows the superiority of reasoning over database querying: the fact that the Piccante pizza is non-vegetarian is not explicitly written in the ontology. It has been inferred from the fact that Piccante pizza contains salami, which is meat.

Imagine now that a client is allergic to pepper. She asks, "is there any pepper in the Piccante pizza?"

This time the SPARQL query will be

```
query = sparql.prepareQuery("""
           PREFIX : <http://www.fluxus-virus.com/ontologies/pizza#>
           SELECT ?p
           { ?p rdf:type :Pizza .
           ?p :contains :Pepper . }
    """)
```

i.e., “Which pizzas contain pepper?” The result will again be pizza Piccante.

This time, the reasoner used the transitivity of the `contains` role: since the pizza Piccante contains salami, salami contains pepper, and the role is transitive, we must also have that the pizza contains pepper.

11.2 Conceptual Graphs

This section will present an approach to knowledge representation and reasoning alternative to the Semantic Web. Inspired by Peirce’s *existential graphs* (182), Sowa [20] introduced the concept of *conceptual graph* in 1976. It was developed and enriched by Chein & Mugnier [10] and others afterward. Sowa started a yearly workshop in 1986, which became an international conference (International Conference on Conceptual Structures[8]) in 1993. In 2023, there have already been no less than 28 editions of this conference, testifying that this research area is very active.

As defined by Chein & Mugnier, *conceptual graphs* are based on a fragment of FOL, namely FOL formulas using only the conjunction connective and the existential quantifier [10, p. 92]. In particular, there is no negation operation that would allow us to obtain disjunctions and universal quantifiers. The idea is that predicates and predicate arguments become vertices of an undirected acyclic bipartite graph ☞ 209 (the two partitions being predicates and predicate arguments), in which edges are labeled by numbers indicating argument positions. Here is an example we will use throughout this section: in Fig. 11.6, we see the picture of a boy (Carl) driving a toy car with his pet, chimpanzee Baron.

Fig. 11.6 Carl Detroy driving a toy car together with his pet chimpanzee Baron, in Laindon, Essex, on June 18th, 1949 (Photo by Charles Hewitt/Picture Post/Hulton Archive/Getty Images)

[8] https://iccs-conference.org/?page_id=431

This situation can be represented in FOL as follows:

$$\exists X \mathsf{car}(X) \wedge \mathsf{boy}(\mathsf{Carl}) \wedge \mathsf{chimp}(\mathsf{Baron}) \wedge \mathsf{has\_pet}(\mathsf{Carl}, \mathsf{Baron}) \\ \wedge \mathsf{plays\_with}(\mathsf{Carl}, X) \wedge \mathsf{plays\_with}(\mathsf{Baron}, X) \wedge \mathsf{drives}(\mathsf{Carl}, X),$$

meaning that there is a car X we don't wish to name; that Carl is a boy and Baron a chimp; that Baron is Carl's pet; that both Carl and Baron play with the car; and that Baron is driving it (since he is sitting on the front seat).

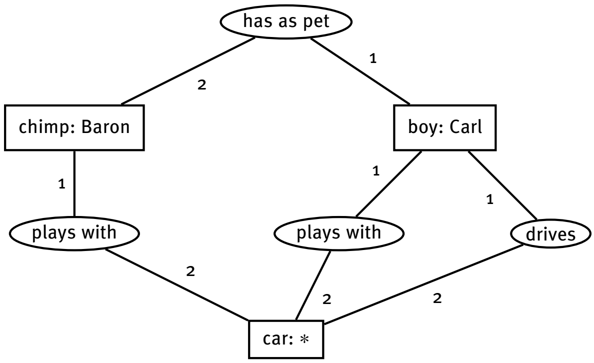


Fig. 11.7 A CG describing Fig. 11.6

In the graph of Fig. 11.7, oval boxes represent predicates, square boxes represent their arguments, numbers represent the argument's position, `boy`, `chimp` are "classes" and `Baron`, `Carl` instances of these classes. Not only does this graph show the relations between the different actors of the situation, but it has a very nice property: logical assumption is equivalent to graph homomorphism.

Let us consider the conceptual graph (from now on CG) approach in more detail.

11.2.1 Definitions

To establish the link between the domain we want to describe and the graph structure, we define the notion of *vocabulary*. A *conceptual graph vocabulary* [10, pp. 22–23] is a triplet $(T_C, T_R, \mathcal{I})$, where

- T_C is the set of *concept types* (our "classes"), it is partially ordered by a relation $\geq$ (hyperonymy), for which the greatest element is $\top$;
- T_R is the set of *relation symbols*, it is also partially ordered by a relation $\geq$ (hyperonymy of relations) and each element has an *arity* (a positive nonzero integer);
- $\mathcal{I}$ is the set of *individual markers* (instances). We add to $\mathcal{I}$ the symbol $*$ (*generic marker*) with the property $\forall i \in \mathcal{I}, i \geq *$. We call $\mathcal{M} = \mathcal{I} \cup \{*\}$ the set of *markers*.

These sets are mutually disjoint. Having a generic marker, we can make assertions involving an arbitrary element in a class without using the $\forall$ quantifier, which is not part of the current FOL fragment.

A *basic conceptual graph* (BCG) [10, pp. 25–27] on a vocabulary $(T_C, T_R, \mathcal{I})$, is an undirected bipartite graph G, with partitions C and R, with potentially multiple edges[9] and a *labeling function* ℓ, defined as follows:

- vertices are *concepts* C and *relations* R, these are the two partitions of the bipartite graph;
- the image of a concept c by ℓ is a pair (t, i) with $t \in T_C$ and $i \in \mathcal{I}$ (a concept node is labeled by a class and a marker, either individual or generic);
- for $r \in R$, $\ell(r) \in T_R$;
- the degree of a relation node r is equal to the arity of the relation symbol $\ell(r)$ (i.e., each relation node has precisely as many neighbors as the arity of the relation symbol, neither more nor less);
- edges incident to a relation node r are numbered from 1 to the degree of r.

Let us now use the partial orders by which the set of concept types and the set of relation symbols are endowed. These correspond to hyperonymic relations between concepts, as in child $\geq$ boy ("child" is more general than "boy," or, similarly, in an interpretation $\mathbf{I}$ of our formula, $\mathsf{boy}^{\mathbf{I}} \subseteq \mathsf{child}^{\mathbf{I}}$), or, between relation symbols, as in is_parent $\geq$ is_ancestor.

Let (t, i) and (t', i') are concepts (concept type and marker), we define an *order of concepts* [10, p. 28] out of the order of concept types by the formula

$$(t, i) \leq (t', i') \equiv (t \leq t') \wedge (i \leq i').$$

Note that this definition is misleading because the order $i \leq i'$ is *partial* and can be asserted *only in very specific cases*. As markers are individual instances of concepts, they actually *cannot* be compared, *except* if one of them is the generic marker (or if we have equality). So, the only case where the comparison $i \leq i'$ makes sense is when $i = i'$ or $i = *$, in which case i' can be arbitrary. Let us restate the definition of concept order accordingly:

$$(t, i) \leq (t', i') \equiv \big((t \leq t') \wedge (i = i')\big) \vee \big((t \leq t') \wedge (i = *)\big).$$

11.2.2 Subsumption

Recall from algebra that an *homomorphism* between two structures is a map respecting internal operations (e.g., $\varphi(a + b) = \varphi(a) + \varphi(b)$, $\varphi(\lambda a) = \lambda\varphi(a)$, etc.).

In our case, if G and G' are conceptual graphs on the same vocabulary, an *homomorphism of conceptual graphs* is an application φ:

---

[9] This case arises when a predicate has two or more identical arguments, as in $P(A, A)$. We should normally call such a graph a *multigraph*, but we chose not to make the terminology more cumbersome.

- sending edges to edges, i.e., when there is an edge e between v_1 and v_2, then there is an edge between $\varphi(v_1)$ and $\varphi(v_2)$, namely $\varphi(e)$;
- respecting partition, i.e., sending C into C', and R into R',
- sending $\mathcal{I} \cup \{*\}$ into $\mathcal{I}' \cup \{*\}$, in a way such that $\varphi(c) \leq c$, $\varphi(r) \leq r$, and
- respecting edge numbers.

What is essential in this definition is that $\varphi(c) \leq c$ and $\varphi(r) \leq r$. These inequalities mean that we can send a concept or a relation either to itself or to a hyponym, i.e., we can send child:$*$ to child:Carl, or child:$*$ to boy:$*$, or child:$*$ to boy:Carl, but never boy:$*$ to child:$*$, etc.

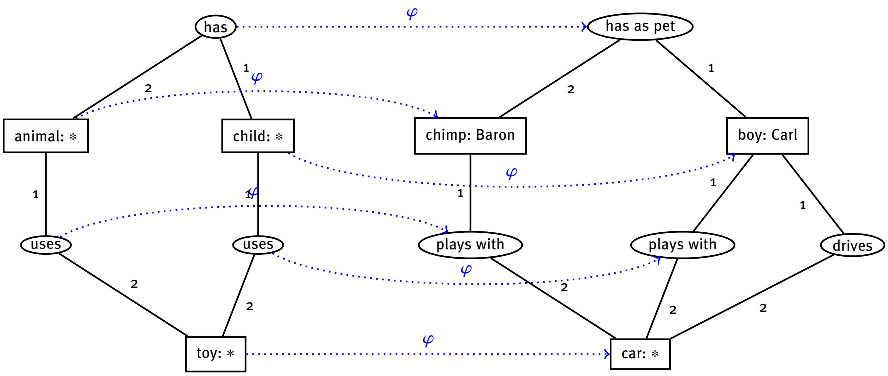


Fig. 11.8 An homomorphism φ between two CGs

In the example of Fig. 11.8 we have indeed a homomorphism φ, since (chimp:Baron) $\leq$ (animal:$*$), (boy:Carl) $\leq$ (child:$*$), (car:$*$) $\leq$ (toy:$*$), has_as_pet $\leq$ has, plays_with $\leq$ uses, plus all the other necessary properties of a homomorphism (such as respecting edge numbers, etc.).

Let G and G' be conceptual graphs on the same vocabulary. We define the *subsumption relation* $\geq$ [10, p. 31] as follows: $G \geq G'$ iff there exists a conceptual graph homomorphism $\varphi \colon G \to G'$. Intuitively, this means that the situation described by G' is a special case of the situation described by G. Indeed, the structures are compatible, and every node of G' is a special case of the corresponding node of G.

11.2.3 Queries

Imagine having a vast depository of images described using CGs over the same vocabulary. Wouldn't it be nice if one could search images in that depository using similar descriptions? Therefore, A CG can be considered a *query*. We will say that a query Q is *accepted* by some graphs G_i if for every G_i we have $Q \geq G_i$. And we get even better: by looking at the image of $\varphi_i(Q)$ inside each G_i we can see *why* G_i accepts Q.

Let Q and G be conceptual graphs on the same vocabulary. We will say that Q is a query accepted by G if $Q \geq G$. The query results are the images $\varphi(Q)$ for each homomorphism $\varphi\colon Q \to G$.

Let us take, for example, the sentence “A child drives a car” as a query. The corresponding CG can be seen on the left side of Fig. 11.9.

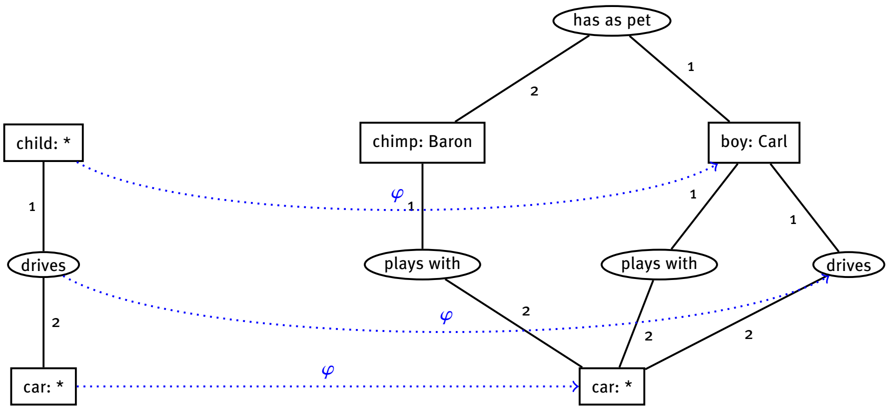


Fig. 11.9 The query Q “A child drives a car” is accepted by G.

To get an explanation of the reasons why the CG accepts this query, let us look at $\varphi(Q)$: Carl is a specific boy; therefore he is a good candidate for an arbitrary child; the verb drives is the same in both CGs, and so is the concept of car. Are there other ways our CG could accept Q? Actually not: a chimp is not a kind of child (at least for most of us—some people may think differently), and playingwith is not a kind of driving, it is instead the opposite that is true (driving is one of the many ways one can play with a (toy) car, the opposite is not true).

Now, try a different kind of query: let us search for pictures showing a child and an animal. The CG representing this query can be seen on the left side of Fig. 11.10; it consists of two unconnected square boxes for the concepts animal:∗ and child:∗.

We see in Fig. 11.10 that the query is accepted by our CG, by sending child:∗ to boy:Carl and animal:∗ to chimp:Baron.

What about the query “A chimp drives a car”?

As seen in Fig. 11.11, the CG does not accept this query. The reason is immediately visible: the only path of length 2 connecting the chimp Baron with a car goes through the relation plays_with, which is not a hyponym of drives as we already observed. And we cannot send Chimp:∗ to boy:Carl either, so there is no solution.

We have a similar result for the query “A car drives a child.” As seen in Fig. 11.12, if we use the same mapping as in Fig. 11.8, we do not get a CG homomorphism since edge numbers are not respected.

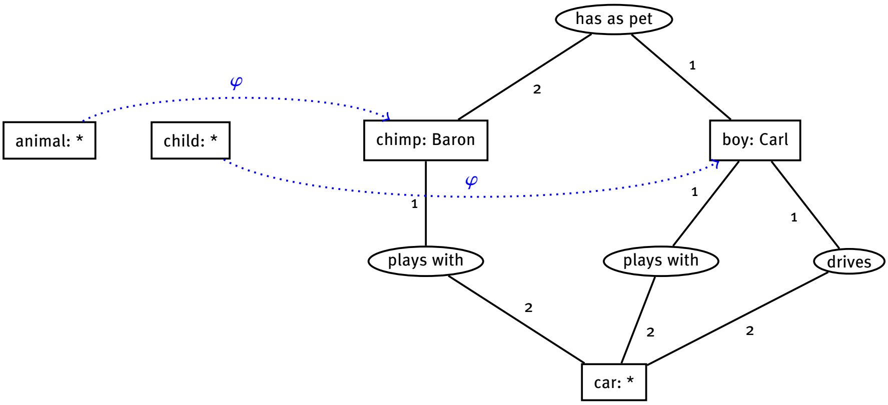


Fig. 11.10 The query Q "There is an animal and a child" is accepted by G.

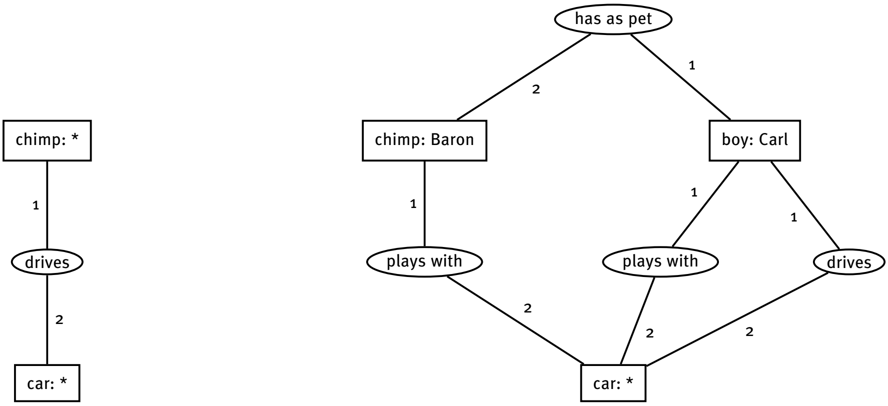


Fig. 11.11 The query Q "A chimp drives a car" is not accepted by G.

These examples show that this kind of search is far more precise than a simple keyword-search of image descriptions.

11.2.4 Beyond CGs: SGs, Models, Translation into FOL

Chein & Mugnier extend the notion of BG by adding *coreference*, in the sense that different markers (not necessarily individual) can refer to the same entity. Without entering into the details, a *simple conceptual graph* (SG) is a BG together with a coreference equivalence relation coref [10, p. 67]. And, as always in mathematics, when we have an equivalence relation, we can take the quotient of the SG by the coref relation. We then obtain the *normal form* of an SG [10, pp. 71–72].

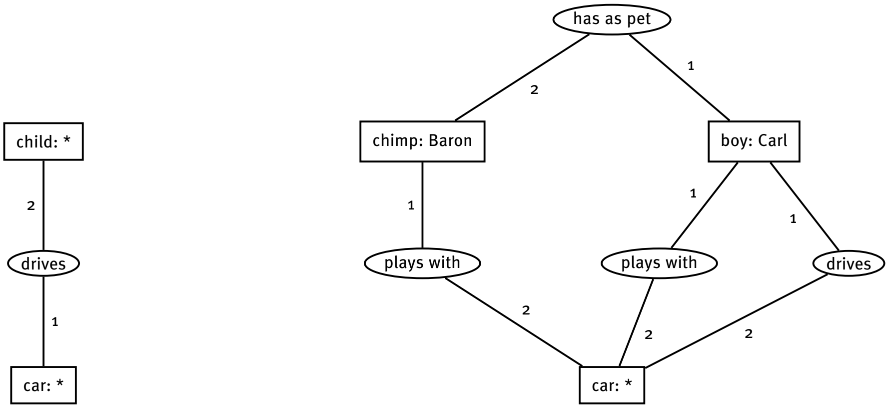


Fig. 11.12 The query Q "A car drives a child" is not accepted by G.

Coreference is important because we want to send coreferent concepts to the same domain elements in any interpretation. In the following, we will consider only interpretations that are true in their domain, that is, *models*.

11.2.4.1 Models

Let us define models of SGs [10, pp. 84–85]:

- let $\mathcal{V} = (T_C, T_R, \mathcal{I})$ be a vocabulary, a model of $\mathcal{V}$ is a pair (D, δ) where D is the domain and δ is a mapping such that:
  - for every concept type t_c, $\delta(t_c)$ is a subset of D such that if $t_c \leq t'_c$ then $\delta(t_c) \subseteq \delta(t'_c)$,
  - for every relation symbol t_r, $\delta(t_r)$ is a subset of $D^{\mathsf{arity}(t_r)}$ such that if $t_r \leq t'_r$ then $\delta(t_r) \subseteq \delta(t'_r)$,
  - for every individual marker i, $\delta(i) \in D$;
- let $(C, R, E, \ell, \mathsf{coref})$ be an SG over a vocabulary $\mathcal{V}$, and (D, δ) a model of the vocabulary, then we define an *assignment* α as a mapping from C to the domain D, such that if c_1 and c_2 are coreferent then $\alpha(c_1) = \alpha(c_2)$ in D;
- let $G = (C, R, E, \ell, \mathsf{coref})$ be an SG over a vocabulary $\mathcal{V}$ and (D, δ, α) a triple consisting of a model (D, δ) of the vocabulary and an assignment α. The triple (D, δ, α) is a *model satisfying* G if the assignment α is such that:
  - when a concept c has an individual marker i, then $\alpha(c) = \delta(i)$, i.e., in the vocabulary's model, we have already assigned an element of the domain to each (individual) marker, now we send all concepts having that marker to that very same element,
  - more generally, $\alpha(c) \in \delta(t_c)$, i.e., the assignment of a concept must fall into the subset of D that corresponds to the concept's type,

– for every relation r, if $c_1, \ldots, c_k$ is the list of neighbors of r (corresponding to the elements of the predicate in FOL), where k is of arity t_r, then $(\alpha(c_1), \ldots, \alpha(c_k)) \in \delta(t_r) \subseteq D^{\mathsf{arity}(t_r)}$.

Let us explain these notions by taking again the example of the CG of Fig. 11.7, which has no coreferent concepts and is, therefore, also an SG. Let us call this graph G. The vocabulary of G is car, boy, chimp (concept types), has_as_pet, plays_with, drives (relation symbols), Baron, Carl, $*$ (markers).

Let us take as D the real world as it was on June 18th, 1949 (the day the picture was taken). The interpretation $\delta(\mathsf{car})$ will be the set of all cars; $\delta(\mathsf{boy})$, the set of all boys; $\delta(\mathsf{chimp})$, the set of all chimpanzees. Furthermore, $\delta(\mathsf{has\ as})$ will be the set of all pairs of people and their pets; $\delta(\mathsf{plays\_with})$ the set of all pairs of those (people or animals) playing with something, and the object they are playing with; $\delta(\mathsf{drives})$ the set of all pairs of those (people or animals) driving something, and the object they are driving. Finally, $\delta(\mathsf{Carl})$ will be Carl Detroy, and $\delta(\mathsf{Baron})$ will be Carl Detroy's chimpanzee called Baron.

Let us now take concepts: pairs of concept types and markers. The assignment of concept $\alpha((\mathsf{chimp}, \mathsf{Baron}))$ will be $\delta(\mathsf{Baron})$; $\alpha((\mathsf{boy}, \mathsf{Carl}))$ will be $\delta(\mathsf{Carl})$, and $\alpha((\mathsf{car}, *))$ will be some element of $\delta(\mathsf{car})$, which is the set of all cars.

Finally, for the interpretation to be true and, therefore a model, we must have the following:

- the assignments of the neighbors of the node of the has_as_pet relation, namely $(\alpha((\mathsf{chimp}, \mathsf{Baron})), \alpha((\mathsf{boy}, \mathsf{Carl})))$, that is $(\delta(\mathsf{Baron}), \delta(\mathsf{Carl}))$, must belong to $\delta(\mathsf{has\_as\_pet})$, which is the set of pairs of people having a pet and their pets. In other words, it must be true that Carl Detroy belongs to the people who have a pet, and Baron must be one of his pets. And this is indeed the case by construction;
- and similarly for the other three relations.

11.2.4.2 Translation into FOL

Conceptual graphs are interesting because SG homomorphism is equivalent to deduction, and hence consequence, in FOL. To understand how this works, let us define a standard way for translating SGs into FOL formulas [10, pp. 89–91]. Obviously, relation nodes of degree k will be sent to predicates of arity k, but there is much more information hidden in an SG, which we will explore in this section.

Let us first consider the vocabulary. Every concept type corresponds to a unary predicate using the same name, and every relation symbol of arity k to a k-ary predicate using the same name. Every individual marker corresponds to a constant using the same name.

This is not all the information we extract from a vocabulary. In fact, for every pair of concept types t_c and t'_c such that $t_c \le t'_c$ we have a FOL formula

$$\forall x\ t_c(x) \to t'_c(x),$$

and for every pair of relation symbols t_r and t'_r of arity k, such that $t_r \leq t'_r$ we have a FOL formula

$$\forall x_1 \cdots \forall x_k \; t_r(x_1, \ldots, x_k) \rightarrow t'_r(x_1, \ldots, x_k).$$

If $\mathcal{V}$ is the vocabulary, then let us denote by $\Phi(\mathcal{V})$ the conjunction of all formulas obtained from pairs $t_c \leq t'_c$ and $t_r \leq t'_r$.

Let us now translate an SG into FOL. We will do it stepwise, starting with concepts. Here are the steps:

1. for each set of coreferential concepts $X = \{c_1, \ldots, c_k\}$, we have one of two things:
   - either this set contains an individual marker i, in this case, we assign to each concept of the coreference set the same term $\hat{c}$, which will be the image of i in the translation of the vocabulary,
   - or it contains only generic markers. In this case, we assign to each concept c of X the same variable $\hat{c}$, which has to be different for each coreference set;
2. let c be a concept node of type t_c, we assign to c the predicate $t_c(\hat{c})$;
3. let r be a relation node of type t_r and arity k, we assign to r the k-ary predicate $t_r(\hat{c}_1, \ldots, \hat{c}_k)$, where $c_1, \ldots, c_k$ are the neighbors of r in the SG;
4. we take the conjunction ϕ of the predicates obtained from all concept and relation nodes of the SG;
5. $\Phi(G)$ will be the *existential closure* of this conjunction, i.e., we add an existential quantifier for every free variable of ϕ.

Let us see how to apply this process to translate the SG of Fig. 11.7 into FOL. First, the vocabulary: car, boy, chimp, (concept types) become unary predicates; has_as_pet, plays_with, drives, (relation symbols) become binary predicates; Baron and Carl become constants.

There are no inequalities among this SG's concept types or relation symbols. Therefore $\Phi(\mathcal{V})$ is empty.

Let us now apply the steps:

1. the coreference sets are singletons, we have the three concepts (chimp, Baron), (boy, Carl) and (car, $*$). For the first two, we have individual markers; therefore, the terms we get will be simply the constants Baron and Carl. For the third concept, we have a generic marker, and therefore, the term we obtain will be a variable X;
2. based on the previous step, we assign to the three concept nodes (chimp, Baron), (boy, Carl) and (car, $*$), the predicates chimp(Baron), boy(Carl) and car(X);
3. to the four relation nodes has_as_pet, plays_with (on the left), plays_with (on the right) and drives, we assign the predicates has_as_pet(Baron, Carl), plays_with(Baron, X), plays_with(Carl, X) and drives(Carl, X);
4. we have ϕ the conjunction of all the formulas of steps 2 and 3;
5. we set $\Phi(G) = \exists X \; \phi$, since X is the only free variable of ϕ, and we are done.

Here is the full formula:

$$\Phi(G) := \exists X\, \mathsf{chimp}(\mathsf{Baron}) \wedge \mathsf{boy}(\mathsf{Carl}) \wedge \mathsf{car}(X) \wedge \mathsf{has\_as\_pet}(\mathsf{Baron}, \mathsf{Carl}) \\ \wedge\, \mathsf{plays\_with}(\mathsf{Baron}, X) \wedge \mathsf{plays\_with}(\mathsf{Carl}, X) \wedge \mathsf{drives}(\mathsf{Carl}, X).$$

Now that we have seen how the translation from SGs to FOL is obtained, here is an important theorem that establishes SGs as a valid logical theory based on the conjunctive existential fragment of FOL:

THEOREM ([10, p. 92]). – *Let G and H be two SGs over the same vocabulary $\mathcal{V}$ with H in normal form. There is a homomorphism $G \to H$ iff $\Phi(\mathcal{V}) \wedge \Phi(H) \models \Phi(G)$.*

11.3 Exercises

Exercise 10-1: Translate DL formulas into Attempto Controlled English and find counterexamples

The following formulas are inspired by Frank Wolter's Comp 312 course at the University of Liverpool:

$$\exists\mathsf{manages}.\mathsf{Project} \sqsubset \mathsf{Manager}$$
$$(\geq 7\mathsf{manages}.\mathsf{Employee}) \sqsubseteq \mathsf{Manager}$$
$$\exists\mathsf{manages}^{-}.\mathsf{Manager} \neq \bot.$$

Translate them into English. Find a counterexample for each formula and add the corresponding formulas to the original one. Translate them into Attempto Controlled English. Inspect paraphrases and create drawings with *owl-cli*. Prove that the set of formulas (original formula + counterexample) is inconsistent.

Exercise 10-2: Write a Q&A system for the Wildlife Ontology

In § 10.3.2, we have described AWO, the African Wildlife Ontology. In this exercise, we will expand it and build a Q&A system that will communicate with it through *owlready2* and SPARQL queries.

Here are the contents of our ontology in English:

Animals and plants form classes.
Trees are a type of plant.
Branches are parts of trees.
Carnivores are exactly those animals that also eat animals.
Leaves are parts of branches.
Giraffes are herbivores and eat only leaves.
Herbivores are animals that eat only plants or parts of plants.
Lions are animals that eat only herbivorous animals.

Giraffes and lions are disjoint classes.
Plants are a disjoint class from animals.
Leo is a lion.
Gigi is a giraffe.
Giginou is a giraffe, different from Gigi.

Translate these sentences into Attempto to obtain an ontology through Attempto's parser.

Analyze the following three question patterns:

1. *Who are the* (put here a class name, in the plural)*?*
   Expected answers: "There is no (class name).", "There is a single (class name). It is (name of individual)." or "There are (quantity) (class name in plural). They are (list of individuals)."
2. *Is* (name of individual) *a* (class name)*?*
   Expected answers: "Yes" or "No."
3. *What do we know about* (name of individual)*?*
   Expected answers: "(name of individual) is a (name of class) which is a (etc., the complete hierarchy)."

Build a Q&A system that will analyze the answer, send the appropriate query, retrieve the result, and answer to the user.

Test your system on the following nine questions:

Who are the giraffes?
Is Léo a lion?
What do we know about Gigi?
Tell me, who are the bloody lions?
Is Gigi actually a giraffe?
What is there to know about Léo?
Who are the whales?
What do we know about Davros?
Sorry for bothering you, but is Giginou, by any chance, a Dalek?

Be sure to provide the answer "There is no X in AWO" if the individual or the class asked is not part of the AWO.

References

[1] Carlos Buil Aranda et al. "SPARQL 1.1 Overview". https://www.w3.org/TR/sparql11-overview/. 2013.
[2] David Beckett et al. "RDF 1.1 Turtle". https://www.w3.org/TR/turtle/. 2014.
[3] Tim Berners-Lee. "Information Management: A Proposal". https://www.w3.org/History/1989/proposal.html. 1989.
[4] Tim Berners-Lee, Roy Fielding, and Larry Masinter. "Uniform Resource Identifier (URI): Generic Syntax". In: *Request for Comments* 3986 (2005).
[5] Tim Berners-Lee et al. "A Framework for Web Science". In: *Foundations and Trends in Web Science* 1 (2006), pp. 1–130.
[6] Paul V. Biron, Kaiser Permanente, and Ashok Malhotra. "XML Schema Part 2: Datatypes Second Edition". https://www.w3.org/TR/xmlschema-2. 2004.

[7] Piero A. Bonatti and Luigi Sauro. “Sticky Policies in OWL2: Extending PL with Fixpoints and Transitive Closure”. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*. Aug. 2022, pp. 73–80.

[8] Neil Bradley. *The XML Companion*. 3rd ed. Pearson Business, 2001.

[9] Dan Brickley and R.V. Guha. “RDF Schema 1.1”. https://www.w3.org/TR/rdf-schema/. 2014.

[10] Michel Chein and Marie-Laure Mugnier. *Graph-based Knowledge Representation. Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing Series. Springer, 2009.

[11] Pradeep Dantuluri, Brian Davis, and Siegfried Handschuh. “A Use Case for Controlled Languages as Interfaces to Semantic Web Applications”. In: *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10)*. Valletta, Malta: European Language Resources Association (ELRA), May 2010.

[12] Martin Dürst and Michel Suignard. “Internationalized Resource Identifiers (IRIs)”. In: *Request for Comments* 3987 (2005).

[13] John H. Gennari et al. “The evolution of Protégé: an environment for knowledge-based systems development”. In: *International Journal of Human-Computer Studies* 58.1 (Jan. 2003), pp. 89–123.

[14] Charles F. Goldfarb. “The Roots of SGML – A Personal Recollection”. http://www.sgmlsource.com/history/roots.htm. 1996.

[15] Yannis Haralambous and Martin Dürst. “Unicode from a Linguistic Point of View”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 167–183.

[16] Pascal Hitzler et al. “OWL 2 Web Ontology Language Primer (Second Edition)”. https://www.w3.org/TR/2012/REC-owl2-primer-20121211/. 2012.

[17] C. Maria Keet. *An Introduction to Ontology Engineering*. Vol. 20. Computing. College Publications, 2018.

[18] Jean-Baptiste Lamy. *Ontologies with Python: Programming OWL 2.0 Ontologies with Python and Owlready2*. Apress, 2020.

[19] Laurent Prévot et al. “Ontology and the lexicon: a multidisciplinary perspective”. In: *Ontology and the lexicon, a Natural Language Processing perspective*. Ed. by Chu-ren Huang et al. Studies in Natural Language Processing. Cambridge University Press, 2010, pp. 3–24.

[20] John F. Sowa. “Conceptual Graphs”. In: *IBM Journal of Research and Development* 20 (1976), pp. 336–357.

Part III
Data Formats

The task of Natural Language Processing consists in applying the abstract principles of linguistics (Part I of this book) and the mathematical tools used to model them (Parts II and IV) to *data*. We are nowadays so used to digital technologies in our daily life that we forget that the way we represent language on computers is discretized, segmented, and codified. A character string is not a word as we would utter it, whether orally or in written form. A character string *represents* a word in a very specific way, often hidden behind flashy human-machine interfaces. In this part, we will look at technologies that are mostly taken for granted: how a text is stored and how the structure of a document is represented, be it a single sentence or a huge corpus. In this part we will cover the basics: on the lowest level, text in Unicode; then, documents and their general structure, in XML; and finally, corpora, specific XML instances for various document types.

Chapter 12
Unicode

> If a noncharacter is received in open interchange, an application is not required to interpret it in any way. It is good practice, however, to recognize it as a noncharacter and to take appropriate action
>
> *The Unicode Standard Version 15*

Unicode is a computing industry standard for encoding, representing, and handling text. It was introduced in 1991 and is nowadays practically the only text encoding standard worldwide. It introduces itself as being a *character encoding* [2, p. 1]. Nevertheless, there is a caveat: the notion of "character" is specific to computer science and does not exist in linguistics. Let us start this section by reviewing this fundamental notion.

12.1 What Is a Character?

Humans learn to speak in infancy simply by being immersed in speaking communities (family, other children). Contrary to speech, which is learned automatically by simple immersion, writing needs instruction, and instructors will naturally model a writing system to make it digestible. Instructors will build lists of signs, called letters, syllables, sinograms, etc., so they can communicate their scriptural knowledge in an orderly and systematic way. This holds for how linguistic units are written (words, sentences, etc.), called *orthography*. Still, it also holds for defining the basic list of writing units (alphabets, syllabaries, etc.).

In Chapter 3, we defined the notion of *grapheme* as the elementary unit of a writing system. It is defined by a *descriptive* approach: by observing written productions and performing commutation tests, one can extract elementary units, *graphemes*. This is not what happens when writing is taught—a *prescriptive* approach is used instead. Some authority (an instructor, an academy, a ministry, …) decides the elementary units of writing so that inside a given community, written communication remains homogeneous. Let us call the resulting units *orthographemes*. Or- ☞ 45

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_12

thographemes tend to overlap largely with graphemes, but there are exceptions. For example, the Arabic ligature "لا"AR is given a letter status in schoolbooks [4, p. 263], and it also appears in Arabic typewriter keyboards, even though it is not a grapheme.

Orthographemes are used in education and correspond to the perception an arbitrary writer of a language has of her language. But besides reading and writing by hand, other tools have been used to produce and transmit written information. For example, sometimes, information must be represented numerically to be transmitted or stored efficiently. This is, of course, the case of the computer. Still, it occurred already in Antiquity when, according to the historian Polybius (2nd c. BCE), General Aeneas used a 5-bit code for the 24 Greek letters:

> The first letter is κ. This comes in the second group of letters and, therefore, on the second tablet; the signaller, therefore, must raise two torches on the left to show the recipient that he must look at the second tablet; then he will raise five on the right because κ is the fifth letter in the group, which the recipient must thereupon write on his tablet. Then the signaller must raise four torches on the left, for ρ is in that group, and two on the right, because it is the second in the fourth group, and the recipient will write ρ on his tablet: and so on for the other letters.[1] [24, p. 10.46]

At that time, no punctuation was used, but general Aeneas used an "end-of-message" code.

Skipping 23 centuries, the same principle of mapping orthographemes into numeric values is used today in computing. The resulting units are called *characters* in this context.

But orthographemes do not come out of thin air: by definition, they presuppose the existence of an authority that declares them as such. In the case of Unicode, this authority is the *Unicode Consortium*, a non-profit organization founded in 1991 and incorporated and based in Mountain View, CA.

12.2 Unicode Structure and Terminology

Unicode has its vocabulary, which we will briefly present in this section. The numerical values used to encode characters are called *code points* [2, p. 88]. There are 1,114,112 available code points, but only a small part among them is used—the day an extraterrestrial civilization arrives with a 500k grapheme writing system, we will maybe be able to fill the entire Unicode *codespace*.

What is closest to (ortho)graphemes is the notion of *abstract character*, a "unit of information used for the organization, control or representation of textual data" [2,

---

[1] The Greek text is so beautiful that we cannot resist the temptation of giving it in full: "πρῶτον δ᾽ ἐστὶ γράμμα τὸ κάππα· τοῦτο δ᾽ ἐστὶν ἐν τῇ δευτέρᾳ μερίδι καὶ τῷ δευτέρῳ πλατείῳ. δεήσει δὲ καὶ πυρσοὺς ἐκ τῶν εὐωνύμων δύ᾽ αἴρειν, ὥστε τὸν ἀποδεχόμενον γινώσκειν ὅτι δεῖ τὸ δεύτερον πλατεῖον ἐπισκοπεῖν. εἶτ᾽ ἐκ τῶν δεξιῶν ἀρεῖ πέντε, διασαφῶν ὅτι κάππα· τοῦτο γὰρ πέμπτον ἐστὶ τῆς δευτέρας μερίδος, ὃ δεήσει γράφειν εἰς τὸ πινάκιον τὸν ἀποδεχόμενον τοὺς πυρσούς. εἶτα τέτταρας ἐκ τῶν εὐωνύμων, ἐπεὶ τὸ ῥῶ τῆς τετάρτης ἐστὶ μερίδος. εἶτα δύο πάλιν ἐκ τῶν δεξιῶν· δεύτερον γάρ ἐστι τῆς τετάρτης. ἐξ οὗ τὸ ῥῶ γράφει [ὁ δεχόμενος τοὺς πυρσούς]· καὶ τὰ λοιπὰ τὸν αὐτὸν τρόπον."EL

p. 88]. An abstract character has *properties* [2, p. 99], some of which are *normative* and others *informative*. The most important property is the *name*, so, for example, the abstract character name for the uppercase form of the first letter of the Latin alphabet as used in the US is LATIN CAPITAL LETTER A and the names of the abstract characters representing the three Japanese mystic apes Mizaru, Kikazaru and Iwazaru (the first covering his eyes, the second his ears and the third his mouth) [20, p. 6.4], are SEE-NO-EVIL MONKEY, HEAR-NO-EVIL MONKEY and SPEAK-NO-EVIL MONKEY. Names of abstract characters have to be unique and written following specific rules. And if, for any reason, an error is committed when writing a name in the Unicode standard, that error must remain *ad vitam aeternam*, at most one can expect the error to be indicated by a comment[2].

When we combine the notions of abstract character and of coded point, we get *encoded characters*, or *characters* for short. The abstract characters we mentioned above correspond to characters U+0041 LATIN CAPITAL LETTER A U+1F648 SEE-NO-EVIL MONKEY, U+1F649 HEAR-NO-EVIL MONKEY and U+1F64A SPEAK-NO-EVIL MONKEY (the convention is to write a “U,” a plus sign and the hexadecimal uppercased number of the code point).

Characters are stored and transmitted, but as long as humans still exist on this planet, they will also be displayed on devices such as monitors, mobile phones, smart watches, etc. The γ-graph that *renders* a character is called a *glyph*. Unicode is not ☞ 40
encoding glyphs—nevertheless, glyphs are part of what Unicode calls the *identity* of a character [2, p. 85], namely its name and representative glyph in the code charts. If the reader is wondering why the representative glyph is so important—after all, there is no need to draw an “A” to know what U+0041 LATIN CAPITAL LETTER A looks like—she should consider the fact that many character names are hardly informative, for example, the sinogram “語”JA has the name CJK UNIFIED IDEOGRAPH-8A9E,[3] so much for character name semantics.

Unicode characters live in *blocks*, which are arbitrary ranges of subsequent characters, sometimes inspired by legacy encodings, for example, for the blocks of Indian subcontinent writing systems, which are all based on the same legacy ISCII standard. Otherwise, Unicode is subdivided in 17 *planes* of 65,536 (= 2^{16}) characters. The characters in the first plane (called the *basic multilingual plane* or BMP) therefore need only 2 bytes to be encoded and unfortunately, there is still software out there that can only handle BMP and fails for the remaining 16 planes (Egyptian hieroglyphs and cuneiform characters are outside the BMP, and so are, as of Unicode v. 15, no less than 1,874 emoji[4]). ☞ 50

[2] See character U+1D0C5 BYZANTINE MUSICAL SYMBOL FHTORA SKLIRON CHROMA VASIS in the Byzantine Musical Symbols table. The word FHTORA in the name should be FTHORA (“φθορά”EL ⌊wear⌋) as in the thirteen preceding and the four following characters in the same table. Instead of correcting the error, a comment has been added: “a misspelling of “FTHORA” in character name is a known defect.”

[3] We mention this sinogram, which means “language,” because Masato Hagiwara used it to name his company *Octanove*. Hagiwara is the author of a book on neural networks and NLP [6], which we will use in Chapter 15. The name *Octanove* comes from the hexadecimal `8A9E` that gives: `8` → “oct” (“ὀκτώ”EL), `A` → “a,” `9` → “nov” (“novem”LA), `E` → “e”.

[4] http://www.unicode.org/emoji/charts/full-emoji-list.html

12.3 UTF-8

This is not the place to describe the various ways Unicode characters are stored and transmitted at the level of bits and bytes (for that, see [7, pp. 53–70]). We will make an exception for one *encoding form* that happens to be very popular, namely *UTF-8*. Originally invented by the Turing-awarded computer scientist Ken Thompson at Bell Labs in 1992, UTF-8 is a *variable-length encoding* in the sense that depending on the number of bits you need for a given character, you will use one, two, three or four bytes:

	UTF-8			
UTF-32	Byte 1	Byte 2	Byte 3	Byte 4
`00000000 00000000 0xxxxxxx`	`0xxxxxxx`			
`00000000 00000yyy yyxxxxxx`	`110yyyyy`	`10xxxxxx`		
`00000000 zzzzyyyy yyxxxxxx`	`1110zzzz`	`10yyyyyy`	`10xxxxxx`	
`000uuuzz zzzzyyyy yyxxxxxx`	`11110uuu`	`10zzzzzz`	`10yyyyyy`	`10xxxxxx`

☞ 257 In fact, reading text in this encoding form requires a finite-state automaton: depending on the number of initial ‘`1`’s in the initial byte, it will read no additional byte, or one, two, three additional bytes, and take the “content bits” (denoted by `x`, `y` and `z` in the table above), merge them and extract their numerical value. For example, the Cyrillic “ш” code point is 1096, which is greater than 127; therefore, we will need two bytes. By writing

	byte 1	byte 2
Bit position	1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8
Bit value	`1 1 0 1 0 0 0 1`	`1 0 0 0 1 0 0 0`

positions 1 and 2 of the first byte are ones, and position 3 is a zero, meaning that the remaining five bits must be stored, and an additional byte must be read. If this additional byte starts with a one and a zero, then everything is OK (no data corruption), and we can store the remaining six bytes. What we have stored is `10001001000`, and every machine knows that this is exactly 1,096, namely the code point of U+0448 CYRILLIC SMALL LETTER SHA.

This explains why some French Web pages contain text that looks like this: “FrÃ©dÃ©ric, un garÃ§on gÃªnÃ©” (for “Frédéric, un garçon gêné”FR). It is because the browser is convinced that the text is in a one-byte legacy encoding (such as ISO Latin-1) and renders each byte as a stand-alone character.

12.4 Combining Characters and Normalization

One of the ten 10 Unicode Design Principles [2, pp. 14–24] is Convertibility, namely the fact that

> Accurate convertibility is guaranteed between the Unicode Standard and other standards in wide usage as of May 1993. [2, p. 24]

This means all characters of all (wide-usage) pre-1993 encodings have their counterparts in Unicode. Therefore, there are characters for "â," "ê," even for "ŵ" (used in Welsh). But what if we need a "p̂" (p with a circumflex accent)? This character exists in no language known by the author.

To be able to attach any accent or diacritical mark to any character, Unicode created the notion of *combining character* [2, p. 105] (as opposed to *base character*: letters, syllabograms, sinograms, etc.). The idea is that if you append a combining character to a base character, the glyph of the base character is combined in some way with the glyph of the combining character.

When we write a U+0070 LATIN SMALL LETTER P followed by a combining character U+0302 COMBINING CIRCUMFLEX ACCENT, the rendering engine must, normally, display a "p̂" (if this does not happen, then it is the rendering engine's deficiency, not Unicode's). The reader may wonder why one may want to write a "p̂" in the first place. Sometimes fake accents may have special connotative meanings, such as German-like accents on English words produce fake Germanness (as in the San Francisco rock band's name "ĞŘOŤUŠ") [25, p. 271], or Scandinavianness (as in "Häagen-Dazs," which is supposedly Danish, except that there is no umlaut in Danish) [17, p. 177], or when random diacritics on letters allow distinguishing different narrative characters in Damasio's *Les furtifs* [9, pp. 304–307].

Hell is paved with good intentions, and we have just mentioned two of them: having pre-accented characters and combining characters allowing to render accented glyphs. This means that there are two ways to obtain "â": as a *precomposed* character U+00E2 LATIN SMALL LETTER A WITH CIRCUMFLEX or as a *combining character sequence* U+0061 LATIN SMALL LETTER A followed by U+0302 COMBINING CIRCUMFLEX ACCENT. These two are *canonically equivalent* in that they are interchangeable under all circumstances, a programmer's nightmare.

To solve this problem, Unicode defines two normalization forms: *Normalization Form D* (NFD) requires that all characters that be canonically decomposed be decomposed, and *Normalization Form C* (NFC) requires that NFD be first applied. Then, all characters are composed to obtain canonically equivalent character sequences. In other words, NFD is the longest possible version when counting the number of characters, and NFC is the shortest possible version.

The Web Consortium recommends using NFC Unicode characters in UTF-8 encoded form.

12.5 ZWJ and ZWNJ

Even though Unicode encodes characters and not glyphs, sometimes information must be provided on the glyphs that render specific characters. For example, we mentioned in p. 43 the fact that, in German, intermorphemic ligatures have to be broken: compare the rendering of letters "f" and "l" in words "Auflage"$_{\text{DE}}$ and

"Querflöte"$_{\text{DE}}$. In the first case, there is a morpheme boundary between them (the ligature is broken)—in the second case, they belong to the same morpheme (the ligature is intact).

To include this kind of information in text, Unicode provides the U+200C ZERO-WIDTH NON-JOINER character, also known as ZWNJ. In Python, the two German words of our example can be written

```
Auf\u200Clage, Querflöte
```

to ensure their proper rendering. Similarly, one would use a ZWNJ in Turkish to avoid "fi" ligatures since there are distinct "i" (dotted "i") and "ı" (undotted "i") graphemes in Turkish, and a ligature would make them indistinguishable.

There is also a U200D ZERO-WIDTH JOINER character, also known as ZWJ. Its rationale can be expressed as "let the glyphs preceding and following ZWJ take the form they would take if they were connected to some glyph by a ligature." This is especially useful in Arabic, where many τ-graphs are connected by default. When a "ت" τ-graph is followed by a "ن" graph, they both change form and become "تن." If, for some reason, we need to surround the second one with parentheses, with the current font technology, we lose their contextual behavior: "ت(ن)." To keep their contextual forms, one can use the ZWJ character before and after the opening parenthesis to obtain "ت‍(‍ن)". This process is also used as an abbreviation marker in Arabic [8, pp. 154–155].

12.6 Collation

Collation is the way words (or whatever is equivalent to words in a written language) are ordered. In English, this is a straightforward task: "Zelda" comes after "Zazie" and before "Zygons" (which we denote by "Zazie" < "Zelda" < "Zygons"). This is not the case in other languages. For example, in (pre-reform) Spanish, "ch" was considered a digraph, and therefore one would collate "cucaracha"$_{\text{ES}}$ < "chacal"$_{\text{ES}}$. In French, there is a very specific order for words differing only by their accents, as illustrated by the example "cote"$_{\text{FR}}$ < "côte"$_{\text{FR}}$ < "coté"$_{\text{FR}}$ < "côté"$_{\text{FR}}$ (this is called *backward-accent ordering*). In German, one finds "Nagel"$_{\text{DE}}$ < "Nägel"$_{\text{DE}}$ when looking in a dictionary, but "Nägel"$_{\text{DE}}$ < "Nagel"$_{\text{DE}}$ when looking in a phonebook.

This is why the Unicode Consortium has introduced a very sophisticated collation algorithm [27] that allows defining collation rules for languages. This algorithm and rules for many languages are available in the software package *ICU*[5].

[5] https://icu.unicode.org/home

12.7 Language Tags

This section is not related to Unicode, except that Mark Davis, the co-author of the core specification for language tags [23], also happens to be the Unicode Consortium's co-founder and president. According to Davis [23], languages can be represented by *tags* consisting of the following elements (called *subtags*), separated by dashes:

1. the language name, for which we use either two-letter language codes from ISO 639-1 [13] or three-letter language codes from ISO 639-2 [14], ISO 639-3 [15] or ISO 639-5 [16],
2. optionally, the script used in the language's writing system, for which we use four-letter capitalized codes from ISO 15924 [11],
3. optionally, a region code, for which we use two-letter uppercase codes from ISO 3166-1 [12] or three-digit codes from the UN Standard Country or Area Codes for Statistical Use [26].

Examples: `es-ES` = Spanish spoken in Spain, vs. `es-SV` = Spanish spoken in El Salvador; `jp` = Japanese, vs. `jp-Hrkt` = Japanese written in kana; `ber-Tfng-DZ` = Berber written in Tifinagh script in Algeria, etc.

In some cases, we have *extended language subtags*: a two-letter language tag followed by a dash and a three-letter lowercase subtag, as in `zh-yue` (Cantonese Chinese) vs. `zh-cmn` (Mandarin Chinese).

It is also possible to have

4. one or more variant subtags [23, p. 15], which must be at least five characters long (when they begin by a letter) lowercase strings, registered with IANA[6].

The author's favorite variant subtag is `polyton`, as in `el-Gree-GR-polytoni` ("Greek language written in Greek script and in Greece, in the polytonic form"[7]). A case of triple variant subtags is `sl-IT-rozaj-biske-1994`: Resian (a Slovene dialect) in the San/Giorgio/Bila subdialect as spoken in Italy using the 1994 standard orthography.

And finally, one can have two additional special cases [23, p. 16]:

5. extension parts are initiated by a *singleton subtag*, that is a one-letter tag other than `x`, they must follow specific rules;
6. private parts are initiated by the singleton `x`, what follows can be arbitrary.

Notice that while XML and other Semantic Cake technologies use hyphens in language tags, Unix locales [18, pp. 200–204] require underscores: `fr-FR` for the former becomes `fr_FR` for the latter.

[6] https://www.iana.org/assignments/language-subtag-registry

[7] https://www.polytoniko.org/index.php?newlang=en

12.8 Unicode and Python

Provided one has a recent version of Python, there are some nice Python shortcuts to obtain Unicode characters. For example, either of the two following Python strings

```
1 "\U0001F60A"
2 "\N{SMILING FACE WITH SMILING EYES}"
```

should produce the “smiling face with smiling eyes” emoji. On line 1 we use the convention of a lowercase ‘u’ followed by four hex digits (for characters in BMP) or an uppercase ‘U’ followed by eight hex digits (for characters outside the BMP). On the second line, we write the complete Unicode name (which means that to obtain U+1FBA8 we would need 92 characters, since its name is BOX DRAWINGS LIGHT DIAGONAL UPPER CENTRE TO MIDDLE LEFT AND MIDDLE RIGHT TO LOWER CENTRE), to obtain a single character in the text.

For characters in the BMP and therefore its hexadecimal code can be written with four (hex) digits, you can use the `\u1234` notation, as in `\u00E2` for “â” (beware: it is a lowercase ‘u’ followed by four hex digits or an uppercase ‘U’ followed by eight hex digits).

All Python string operations should normally be Unicode-compliant. For example, a Python string length function applied to the string “Δόξα τῷ Θεῷ”$_{\mathrm{EL}}$ should return eleven (Unicode characters) even though twenty bytes are needed to encode the string in UTF-8.

As for collation, one can use the *icu* package (for that, one must first install *ICU* and then the Python package *pyicu*). Here is how to obtain the correct collation in French:

```
1 import icu
2 print(sorted(["côte","côté","cote","coté"]))
3 collator=icu.collator.createInstance(icu.locale('fr_CA'))
4 print(sorted(["côte","côté","cote","coté"], key=collator.getSortKey))
```

which will first produce the standard collation “cote”$_{\mathrm{FR}}$ < “coté”$_{\mathrm{FR}}$ < “côte”$_{\mathrm{FR}}$ < “côté”$_{\mathrm{FR}}$ (which is standard because it is based on the order of code points in Unicode) and after adding the particular collation order will produce the expected “cote”$_{\mathrm{FR}}$ < “côte”$_{\mathrm{FR}}$ < “coté”$_{\mathrm{FR}}$ < “côté”$_{\mathrm{FR}}$. As the reader may have noticed, we used the `fr_CA` locale on line 3 to obtain this collation order—for some reason, backward-accent ordering is included in the French Canadian locale but not in the Metropolitan French one.

12.9 What To Do When a Glyph Is Missing

Imagine we need to include an exotic glyph in our work. If this glyph corresponds to a Unicode character, we only need to find a font containing it. Many fonts are around, and some (like *Google Noto*, which covers more than 150 writing systems as of May 2023) are pretty universal.

But sometimes, the glyph does not correspond to any Unicode character and doesn’t exist anywhere. In that case, we must add it to some font and use it in our applications. We will consider the most straightforward way of adding and using a new glyph: adding it to Unicode's *Private Use Area* (PUA). Like in Area 51 (where the Roswell incident occurred), anything can happen in the PUA. It is a (rather large) block of 6,400 code points (plus another 131,068 code points in planes 15 and 16) where the user can place any glyph and use it by simply calling the corresponding character. Some communities or institutions use the PUA: the MUFI (Medieval Unicode Font Initiative), SIL (Summer Institute of Linguistics), TITUS (Thesaurus Indogermanischer Text- und Sprachmaterialien), etc., so we won't be the only ones using it.

There are a few low-budget font design applications out there: *FontForge* is free, *TypeTool* and *Glyphs Mini* are affordable. We will now illustrate the addition of a new glyph through a PUA character.

The example we have chosen appears in the Japanese translation of the novel *Flowers for Algernon* (1966) by Daniel Keyes, a very popular speculative fiction book translated into 27 languages. In the story, a mentally disabled person, Charlie Gordon, undergoes an operation that increases his IQ dramatically until, in the end, he regresses again into his former condition. In the beginning, when his IQ is still low, he makes many spelling errors, like in this excerpt:

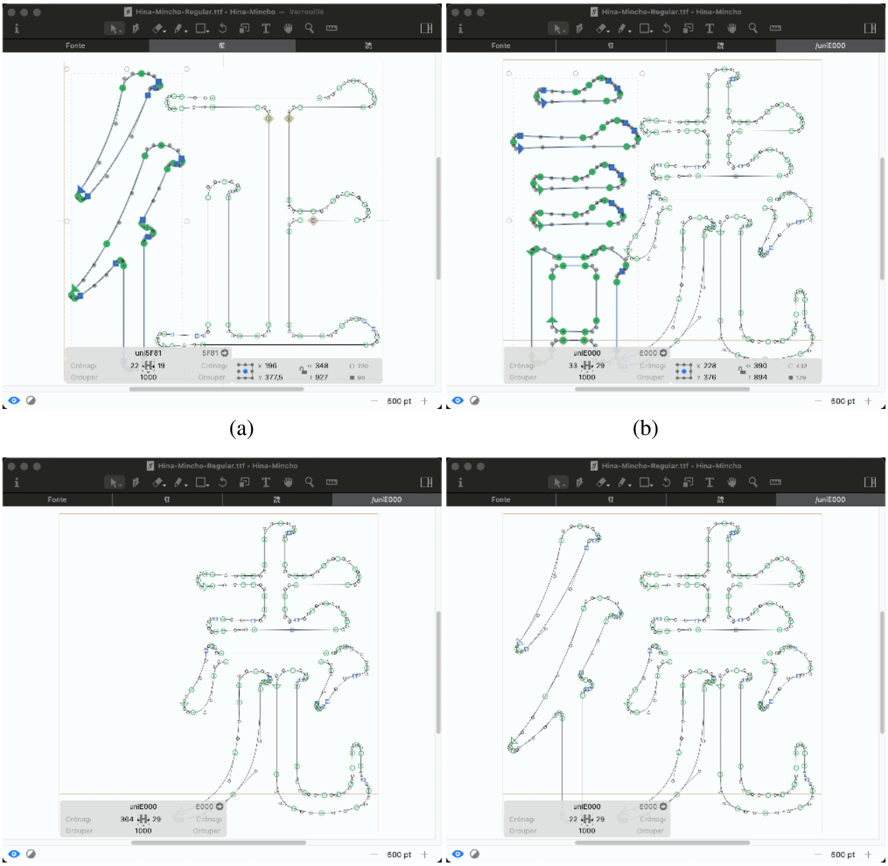
> Burt is very nice and he talks slow like Miss Kinnian dose in her class where I go to lern reeding for slow adults.

Japanese grapheme-phoneme correspondence is much more strict than in English, so the Japanese translator Fusa Obi had difficulty obtaining similar results. One of the things he imagined was sinograms with *wrong components*. To avoid misreadings, ☞ 55 he left the semantic component (the one on the right) intact and changed the phonetic component (the one on the left). He did this for two sinograms only [9, p. 278] because, as he recollects in the book's postface, the publisher was not very keen about adding additional technical difficulties (that was in 1992 when CJKV (Chinese Japanese Korean, Vietnamese) fonts were still out of reach for laymen). The pseudo-sinogram he used most often (14 times) is "彳売," as for the word *"reeding" in the excerpt above. The correct sinogram is "読". Well understood, this glyph exists in no publicly available font, as it is used in a single book only throughout the history of mankind.

To obtain the new glyph, we first found a free Japanese font (opening and modifying a commercial one would be illegal). We have chosen *Hina-Mino-Regular*[8], an "old-fashioned and cute Japanese font," as described by Google. As font-editing software, we use *Glyphs Mini* on the Mac, but other programs exist on many platforms. We open the font with our application and find the glyph of character U+5F81 (to find it, it suffices to type `uni5F81` in the search field on the bottom of the window, as this is the glyph's official name), as can be seen on Fig. 12.1(a), where we have double-clicked on the left part of the glyph. This sinogram will provide us with the left component. We copy the left part into the clipboard.

Then, we search for the glyph of character U+8AAD (the correct version of the misspelled sinogram), select it in the font panel, and duplicate it. We change its

[8] `https://fonts.google.com/specimen/Hina+Mincho`



(a) (b) (c) (d)

Fig. 12.1 Creating a glyph for the non-existing sinogram “” using *Glyphs Mini*

Unicode code point to U+EC7F (or any other code point between U+E000 and U+F8FF). Then we select the left component as in Fig. 12.1(b), remove it as in Fig. 12.1(c), and paste the left component from the clipboard, as in Fig. 12.1(d). We then save the font and use it: the correct sinogram is U+8AAD while the fake sinogram is U+EC7F (in XƎTEX this glyph can be accessed by writing `^^^^ec7f`, in WYSIWYG applications you need to type it using an input tool like *PopChar*). See also §13.3 on how to create the glyph in CDL.

12.10 Further Reading

12.10.1 Literature

Besides Haralambous [7] and Haralambous & Dürst [8], a nice introduction to Unicode is Moran & Cysouw [21]. As for collation, for speakers of German, Küster [19] is an indispensable reference and a very pleasant read.

Suppose the reader is wondering about the "noncharacters" in the epigraph of page 343. In that case, she can rest assured as they are not persecuted victims of discrimination among Unicode characters such as "nonpersons" in real life. There are 66 such noncharacters in Unicode (as of v. 15), mainly for historical reasons, but one of them is very important and, as we will see, has been "sacrificed for the common good." We are referring to noncharacter U+FFFE (it has no name as only characters can have names). To realize the importance of U+FFFE, let us give some background information. When BMP characters are encoded in two bytes, there are two ways of storing/transmitting them: the *big-endian* and the *little-endian* [3]. The terms are from Swift's *Gulliver's Travels* where Big-endians are those breaking their eggs at the big ends, and Little-endians those breaking their eggs at the little ends. These communities are at war, which had already caused 11,000 victims when Gulliver arrived.

To return to Unicode, U+00E9 LATIN SMALL LETTER E WITH ACUTE will be coded `00E9` in big-endian byte order and `E900` in little-endian byte order. As a language, English numbers are big-endian ("twenty-two") while German numbers are (at least partially) little-endian ("zwei und zwanzig"$_{\text{DE}}$). How can software distinguish between big-endian and little-endian byte order? Unicode's solution to this problem is to ask developers to start documents encoded in two-byte BMP Unicode with the character U+FEFF BYTE ORDER MARK. If the document is written in big-endian byte order, then `FEFF` is read as such, and everything is OK; if the document is written in little-endian byte order, then `FFFE` is read, which is a noncharacter. As this cannot happen, the only logical conclusion is that the byte order of the document is little-endian. For this trick to be possible, U+FFFE had to be sacrificed into noncharacterhood, for the sake of all 2-byte encoded documents. This issue does not affect UTF-8 encoded text, which is always big-endian.

12.10.2 LaTeX

To typeset in Unicode, one can use Aleph, XeTeX or LuaTeX. This book has been typesetted in XeTeX. Appropriate fonts can be found online and loaded by the *fontspec* package. The various language-specificities are best handled by the *polyglossia* package.

12.11 Exercises

Exercise 11-1: Steganography Using Unicode Combining Characters

Steganography [5] is the technique of hiding a message (the *payload*) inside data (the *channel*) in a way (the *carrier*) such that the very existence of the message is undetectable. Typically, when the existence of a hidden message is revealed, some third party will interrupt communication, leading to failure to transmit the message. For example, during the Soviet era, Klezmer musicians smuggled information in and outside the Soviet Union by encoding it in musical notation in spiral-bounded notebooks [22]. In this example, the channel was a score, the carrier, a set of symbols (notes, rhythms, articulation indicators, etc.), and the payload, a list of addresses of activists and of people wanting to escape the USSR. Had the KGB agents had the slightest suspicion of the musical nature of the scores, they would have confiscated them, and the operation would have failed.

We will use the choice between composed and decomposed characters to encode textual information. A language with a large density of combining characters is Vietnamese. Our code will be as follows: Vietnamese graphemes with diacritics will be encoded as single Unicode characters or base characters followed by one or two combining characters. An entirely precomposed character will count as an active bit (value 1), and a decomposed sequence of characters will count as an inactive bit (value 0). We will encode uppercase Latin alphabet characters in groups of five bits: values 1 to 26 will be letters “A,” … “Z,” value 0 will correspond to a blank space, values 27 and 28 will correspond to “Message Start” and “Message End”. Values 29, 30, and 31 will be ignored.

In steganographic terminology, our channel will be Vietnamese Unicode-encoded text. The carrier will be the sequence of choices of using entirely composed characters, our sequences of base characters followed by combining characters to represent graphemes, the payload will be a string of uppercase Latin letters and blank spaces starting with a “Message Start,” and ending with “Message End.”

Tasks

Open the Vietnamese translation of the *Universal Declaration of Human Rights*[9] (UDHR) in a browser and copy-paste the text in a file `udhr-viet.txt`. This file will be our channel. Write Python programs that will

1. count the maximum number of characters of the payload string (the *capacity* of the channel);
2. rewrite the channel by hiding the payload string “BAD WOLF” in it;
3. extract the payload from the channel.

[9] `https://unicode.org/udhr/d/udhr_vie.html`

Questions

1. What is the density of this steganographic operation? I.e., how many channel characters are needed to encode a payload character?
2. How many pages of *Moby Dick* can we encode in the Vietnamese *Universal Declaration of Human Rights*? How much Vietnamese data would we need to encode the entire *Moby Dick*? (we mean a simplified "telegraphic" version of *Moby Dick* with only uppercase letters and blank spaces) Would the Vietnamese Wikipedia (as a single file) be sufficient to encode the "telegraphic" *Moby Dick*?
3. Could we combine this approach with compression? How much Vietnamese text would be needed to encode the (not simplified) text of *Moby Dick*, compressed in 7z?[10]
4. How could the *density* (number of payload characters per channel character) be improved?
5. Compare the method of this exercise with the one of Akbas [1, p. 75], who proposes the use of ZWJ and ZWNJ Unicode characters as payload carriers and English language texts as channels. ☞ 347

Exercise 11-2: Southeast-Asian Scripts Implementations in Unicode

South-Asian and Southeast-Asian scripts derive from the Brahmi script (which emerged in the Indian subcontinent in the 3rd century BCE). A genealogy of scripts can be seen on Ben Mitchell's blog[11]. In particular, Thai (U+0E00–0E7F) and Khmer (U+1780–17FF) are close, yet their implementations differ in one important respect. Find out how the implementation of Thai (and of Lao, Tai Le, Tai Viet, etc.) differs from that of Khmer (and Myanmar, Tai Tham, Javanese, Sinhala, etc.).

Exercise 11-3: How do Unicode and Wikipedia deal with simplified and traditional Chinese?

In 1935, the Nationalist Government of China issued the first official list of 324 Simplified Characters. 80% among them were part of the First Draft Simplification Scheme announced by the PRC Ministry of Education in 1956, which introduced 515 simplified characters and abolished 29 characters. In 1964, a General Table of Simplified Characters was published containing 2,236 characters (obtained through the combination of three tables: one of 230 characters for immediate use, one of 285 characters for trial use, and one of 54 components). In 1977, there was an aborted second simplification scheme, and finally, in 1986, an updated scheme was issued by the Chinese State Language Affairs Commission, which is still in use today. It contains 2,234 characters. [10, pp. 35–42]

---

[10] https://7-zip.org/7z.html

[11] https://perma.cc/3735-PB24

Meanwhile, Taiwan and Hong Kong still use traditional, non-simplified Chinese characters.

How does Unicode handle simplified and traditional Chinese characters? Are they merged, as glyphs representing the same character (like 'a' and 'ɑ' both representing LATIN SMALL LETTER A), or are they distinct? And if they are distinct, do they belong to different tables?

And what about Wikipedia? Are there two Chinese Wikipedias, a traditional and a simplified one? After all, the political tensions between the PRC and Taiwan would amply justify the co-existence of two Chinese Wikipedias.

Exercise 11-4: Unicode and political correctness in the processing of emoji

Emoji depict humans and, therefore, run the risk of carrying a politically incorrect message. How does Unicode deal with this issue?

Exercise 11-5: Curiosa Unicodensis

Answer the following questions:

1. what are the semantics of the U+2061 FUNCTION APPLICATION, U+2062 INVISIBLE TIMES, and U+2063 INVISIBLE SEPARATOR characters?
2. why are there "fullwidth" Latin letters, digits, and punctuation (U+FF01–FF5E)?
3. what are high and low surrogates?
4. what is a GETA MARK U+3013 〓?
5. what is the "soft dotted property," how do we get an 'i̇̃' instead of a 'ĩ,' as needed in Lithuanian, and how does Unicode deal with the Turkish 'ı/I' and 'i/İ' letters?

References

[1] E. Ali Akbas. "A New Text Steganography Method By Using Non-Printing Unicode Characters". In: *Eng. & Tech. Journal* 28 (2010), pp. 72–83.

[2] Deborah Anderson et al., eds. *The Unicode Standard, Version 15.0*. Unicode Consortium, 2022.

[3] Danny Cohen. "On Holy Wars and a Plea for Peace". In: *IEEE Computer* 14 (1981), pp. 48–54.

[4] Joseph Dichy. "On the Writing System of Arabic: The Semiographic Principle as Reflected in Nasẖī Letter Shapes". In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 257–268.

[5] Jessica Fridrich. *Steganography in Digital Media*. Cambridge University Press, 2009.

[6] Masato Hagiwara. *Real-World Natural Language Processing*. Manning, 2021.

[7] Yannis Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O'Reilly, 2007.

[8] Yannis Haralambous and Martin Dürst. “Unicode from a Linguistic Point of View”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 167–183.

[9] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. “Graphemic and Graphetic Methods in Speculative Fiction”. In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[10] Kei Imafuku. “Contrasting Approaches to Chinese Character Reform: a Comparative Analysis of the Simplification of Chinese Characters in Japan and China”. PhD thesis. University of Hawai‘i, 2012.

[11] International Organization for Standardization. *ISO 15924:2004. Information and documentation – Codes for the representation of names of scripts*. 2004.

[12] International Organization for Standardization. *ISO 3166-1:2006. Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*. 2006.

[13] International Organization for Standardization. *ISO 639-1:2002. Codes for the representation of names of languages – Part 1: Alpha-2 code*. 2002.

[14] International Organization for Standardization. *ISO 639-2:1998. Codes for the representation of names of languages – Part 2: Alpha-3 code*. 2002.

[15] International Organization for Standardization. *ISO 639-3:2007. Codes for the representation of names of languages – Part 3: Alpha-3 code for comprehensive coverage of languages*. 2007.

[16] International Organization for Standardization. *ISO 639-5:2008. Codes for the representation of names of languages – Part 5: Alpha-3 code for language families and groups*. 2008.

[17] Dejan Ivković. “Towards a semiotics of multilingualism”. In: *Semiotica* 207 (2015), pp. 89–126.

[18] Michael Kerrisk. *The Linux Programming Interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.

[19] Marc Wilhelm Küster. *Geordnetes Weltbild. Die Tradition des alphabetischen Sortierens von der Keilschrift bis zur EDV. Eine Kulturgeschichte*. De Gruyter, 2011.

[20] Patrick Lafcadio Hearn (aka Πατρίκιος Λευκάδιος Χέρν, aka 小泉八雲). *Glimpses of unfamiliar Japan*. Houghton, Mifflin, 1894.

[21] Steven Moran and Michael Cysouw. *The Unicode Cookbook for Linguists. Managing writing systems using orthography profiles*. Language Science Press, 2018.

[22] Lily Hay Newman. “How a Saxophonist Tricked the KGB by Encrypting Secrets in Music”. In: *Wired* (2022). https://perma.cc/BHY3-QSDJ.

[23] “Tags for Identifying Languages”. In: *Request for Comments* 5646 (2009). Ed. by Addison Phillips and Mark Davis.

[24] Polybius. *Histories*. translated by Evelyn S. Shuckburgh. Macmillan, 1889.

[25] Jürgen Spitzmüller. “Floating ideologies: Metamorphoses of graphic “Germanness” ”. In: *Orthography as Social Action*. Ed. by Alexandra Jaffe et al. De Gruyter, 2012, pp. 255–288.

[26] United Nations Statistics Division. *Standard Country or Area Codes for Statistical Use*. Sales No. 98.XVII.9. United Nations publication, 1999.

[27] Ken Whistler and Markus Scherer. “Unicode Collation Algorithm”. https://unicode.org/reports/tr10/. 2022.

Chapter 13
XML, TEI, CDL

Nous sommes tous des documents.[1]

Xavier Aimé & Frank Arnould, *Modélisation ontologique & psychologies*

In this chapter, we will consider the *eXtensible Markup Language* (XML) and three of its applications relevant to Natural Language Processing: the *Text Encoding Initiative*, the *Lexical Markup Framework* and the *Character Description Language*.

13.1 XML

What distinguishes a *text editor* (like *BBEdit* for Mac, or *gedit* for Linux, or *Sublime Text* for Windows) from a *word processor* (like *Microsoft Word* or *OpenOffice Writer*) is that the former displays the exact contents of a file. In contrast, the latter displays an interpreted version of a file's contents. For this reason, the former will display mostly raw text documents (including documents in programming languages such as Python or $\TeX$). In contrast, the latter will display documents in a specific format, including formatting, tables, images, etc.

Suppose we want to stick to the text-editor's "show me every character of the document" point of view but do want to include information on the document's structure (paragraphs, titles, footnotes, tables, etc.). In that case, there is only one solution: *explicit markup*.

Explicit markup is based on the unambiguous separation between "contents" and "markup". If there is the slightest risk of confusion between markup and contents, then, as Rick says in *Casablanca*,

> you'll regret it. Maybe not today, maybe not tomorrow, but soon, and for the rest of your life. [6, p. 121]

[1] "We are all documents."

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_13

☞ 343 It is for this reason that XML has established stringent rules: among all Unicode characters, there are three, and only three, that play a unique role in XML: the ‘less-than’ `<`, the ‘greater-than’ `>`, and the ampersand `&`. There are seven types of markup, and any program reading XML is a finite-state automaton with eight states: one for each type of markup and one for text contents. Before enumerating the seven markup types, let us consider the definition of the notion of *XML name*, which will be needed to define them.

13.1.1 XML Names

According to the XML specification [4, § 2.3] an *XML name* is a string of characters,

- starting with a character, which can be a colon, an underscore, and roughly any grapheme in any of the alphabetic, abjad, or abugida writing systems, any syllable in any of the syllabic writing systems, any sinogram, and the entire contents of the higher Unicode planes,
- and continuing with zero, one or more characters as the ones above, a hyphen, a period, a digit, a middle dot, or a combining diacritical mark.

Notice that the constraint of not starting an XML name with a digit goes only for Arabic ASCII digits. Numerals other than the ASCII Arabic digits can be used everywhere, including as the first character.

Intuitively, the constraint is to use any “reasonable” character (any character you would use in some language), but *not* to start the name with a hyphen, a period, or an ASCII digit.

13.1.2 Markup Types

☞ 257 Put yourself in the shoes of the algorithm reading XML data. As an Finite-State Automaton is needed to switch between the eight states (the seven types of markup and textual contents), we only need to give the entry sequence for each state (the one that makes you enter the state) and the corresponding exit sequence (the one that makes you leave the state). By default, we are in a textual content state. Here are the seven other states:

1. *element tags*: the entry sequence is simply `<` and the exit sequence is `>`. Here are some examples: `<body>`, `<a href="#somewhere">`, `<br/>`, etc.
2. *entities*: the entry sequence is `&` and the exit sequence `;`, between the two there can only be an XML name;
3. the *XML declaration*: a tag that has to be at file begin and looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
```

4. the *DOCTYPE declaration*: the entry sequence is `<!DOCTYPE` and the exit sequence `>`, it contains a Document Type Definition (cf. §13.1.5);

5. *comments*: the entry sequence is <!-- and the exit sequence -->, and there cannot be any other double-hyphen inside it. Comments are intended for humans only, for example <!-- I wrote this on May 2nd, 2023, 00:03:29 -->;
6. *processing instructions*: the entry sequence is <? followed by an XML name, and the exit sequence is ?>, they contain code in some language (given by the XML name in the entry sequence). For example, when code like <?php echo 1+1; ?> is read by a PHP engine, it will output “2” instead;
7. *CDATA sections*: the entry sequence is <![CDATA[and the exit sequence is]]>, the (rather seldom) sections can contain any binary character, for example, image or sound data, provided it does not contain the exit sequence.

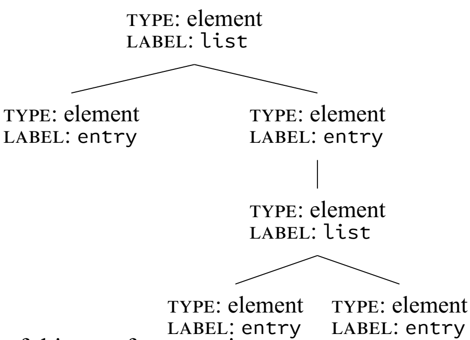
13.1.3 Abstract Syntax Tree Structure

In the previous section, we considered XML documents from the point of view of a program that reads them character by character and switches between the textual contents state and any of the other eight states. In this case, we say that the document is in *serialized* form. But this is only half the story. When reading element tags, an *XML Parser* will build an *abstract syntax tree structure* in memory.

This is the *raison d’être* of XML: to store information into *labeled ordered* tree structures. The nodes of this tree can be of eight different types. Some have a *label*, and some have a *value*.

Here are the eight different types of abstract tree nodes:

1. *elements*: These are the basic building blocks of XML documents. Elements are the only nodes that can be nested; this is how logical structure is represented. For example, the syntax tree of a list (labeled list) of two elements (labeled entry) inside the second entry of a list of two elements will be:



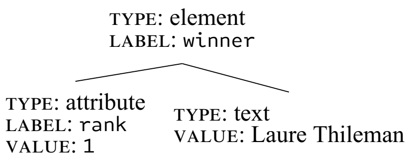
and the serialization of this tree fragment is

```
<list>
    <entry/>
    <entry>
        <list>
            <entry/>
```

```
            <entry/>
        </list>
    </entry>
</list>
```

where we see *opening* tags (no slash), *closing* tags (a slash before the label) and *empty element* tags (the slash after the label).

2. *attributes*: These are label-value pairs, where the label is an XML name and the value is any string (without nested elements) delimited by simple or double quotes. In the XML tree, attributes are inserted before the element's contents, so, for example, the tree of a `winner` element containing an attribute of label `rank` and value `1` and a text fragment "Laure Thileman" would be:



This tree fragment is serialized as follows:

```
<winner rank="1">Laure Thileman</winner>
```

Attributes have no specific order. There cannot be two attributes of the same label in the same element;

3. *namespaces*: we have already discussed namespaces in § 11.1.3. Namespace declarations take the form of attributes, using the prefix `xmlns` followed by a colon and a string, which will be used as the namespace's prefix. The value of that attribute will be the namespace per se. Once a namespace has been declared in some element, the prefix can be applied to elements and attributes anywhere inside the element. For example:

```
<anytag xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <xhtml:table>
        <xhtml:tr>
            <xhtml:td>X</xhtml:td>
        </xhtml:tr>
    </xhtml:table>
</anytag>
```

means that inside the `anytag` element, we use XHTML semantics for a table with a single row and a single cell containing an "X".
The choice of the prefix is arbitrary, but there are conventions. Elements and/or attributes of various namespaces can be mixed. In the abstract syntax tree, namespace nodes immediately follow the element node, in which they are nested, and precede its attribute nodes, if any.

4. *comments* are nodes that can appear anywhere;
5. and so are *processing instructions*,
6. and *CDATA sections*;

7. *text nodes* are always nested in elements. In a text node, to represent the characters `<`, `>` and `&`, one will use the entities `<`, `>`, and `&`. There is another important (meta-)entity: by writing `ሴ`, where `1234` is an arbitrary uppercase hexadecimal number, one obtains the Unicode character U+1234. For example, `😊` will produce the smiling face with smiling eyes emoji;
8. last but not least, there is a unique *document* node, which is the root of the abstract syntax tree. This node has no serialization. It is necessary because, in serialization, there can be comments anywhere, including before the first element opening tag or after the last closing tag. The only way to obtain a tree structure under these conditions is to decide that there is an invisible document node containing everything in the document.

There are two special "hardcoded" attributes:

1. `xml:lang`, by which one can specify the language of a given element (all descendants of this element then inherit the language value). The value of this attribute must be a language tag; ☞ 349
2. `xml:id`, by which one can add a *unique* identifier to any element [16]. The XML Schema mechanism (see §13.1.5) allows one to define many independent families of identifiers, but `xml:id` has unique ID semantics without the need for a schema.

13.1.4 XML Compositionality

Let us consider the following XML document:

```
<?xml version="1.0" encoding="utf-8"?>
<me:statement xmmlns:me="http://www.fluxus-virus.com/me"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    xmlns:wd="http://www.wikidata.org/entity/">
    <xhtml:h1>Some Beloved Places</xhtml:h1>
    <xhtml:ul>
        <xhtml:li><me:place wd:qid="Q207239">Κύθηρα</me:place></xhtml:li>
        <xhtml:li><me:place wd:qid="Q2522847">Shoshone</me:place></xhtml:li>
    </xhtml:ul>
</me:statement>
```

This document's abstract syntax tree structure is displayed in Fig. 13.1.
It uses elements from different namespaces so that for some, the semantics are established in a well-known standard (XHTML), and for others, the semantics are local. Suppose we apply a process similar to Montague formal semantics to attach meaning to the leaves of this tree (the `place` elements) and move upwards. In that case, we realize that the `place` elements are entries of a list, and moving upwards, these entries are merged into a list. At the same time, we have the sentence "Some Beloved Places," which obtains the (XHTML) semantics of a title of level 1 when considering the element in which it is contained, and finally, moving up again, we obtain a `statement` containing the title and the list. By doing this, we obtain the semantics ☞ 167

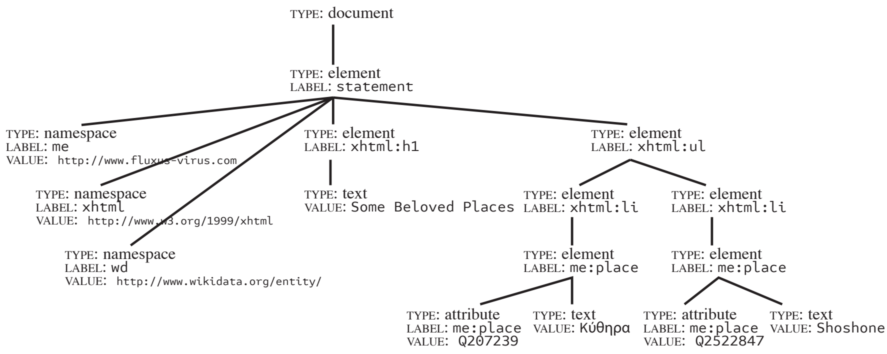


Fig. 13.1 The abstract syntax tree of an XML document with multiple namespaces

of the document by applying the compositionality principle, out of the semantics of the nodes and the way they are ordered in the tree.

13.1.5 XML Schemas

An important feature of XML documents is that they can be *valid* vs. some grammar. This is useful if the XML document is subsequently post-processed by some program. This program will expect a particular structure; otherwise, the post-processing will fail. For example, XHTML is a choice of XML elements and attributes with a specific structure and semantics. A Web browser knows the semantics and expects a given structure to display the XHTML document correctly. For example, suppose you wish to display a table. In that case, you need the cell element to be nested inside the row elements, which will be nested inside the table element—if you do the nesting the other way around, no table will be displayed. To be sure that a given document will be displayed correctly, the first thing to do is to check its *validity* vs. the grammar.

There are several kinds of such grammars: *DTDs* (Document Type Definitions), which have been inherited from XML's ancestor, SGML; *XML Schemas*, which are the official grammars defined by the W3C Consortium; the ISO/IEC 19757-2 standard *RELAX NG* schemas (Regular Languages for XML Next Generation), introduced by OASIS, a Consortium for the promotion of open standards; *Compact RELAX NG* schemas, a much shorter variant of RELAX NG, etc.

An XML Schema is an XML document (in the namespace of XML Schemas) that describes the structure of a family of documents. A document belongs to this family iff it is valid vs. the specific XML Schema. Instead of going into the description of XML Schemas, we will give an example, namely the XML Schema of a simplified version of the document of Fig. 13.1. Here is this document:

```
<?xml version="1.0" encoding="utf-8"?>
<statement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:noNamespaceSchemaLocation="places.xsd">
    <h1>Some Beloved Places</h1>
    <ul>
        <li><place qid="Q207239">Κύθηρα</place></li>
        <li><place qid="Q2522847">Shoshone</place></li>
    </ul>
</statement>
```

This is a simplified version of the document in the sense that no namespaces are used, except for the case of two unique attributes `xmlns:xsi` and `xsi:noNamespace-SchemaLocation`. The latter gives the name of the XML schema file describing the family to which the document must belong, and the former is needed to allow the `xsi` prefix. The schema below is generated by the XML editing software *Oxygen XML Editor*:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3      elementFormDefault="qualified"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5    <xs:element name="statement">
6      <xs:complexType>
7        <xs:sequence>
8          <xs:element ref="h1"/>
9          <xs:element ref="ul"/>
10       </xs:sequence>
11     </xs:complexType>
12   </xs:element>
13   <xs:element name="h1" type="xs:string"/>
14   <xs:element name="ul">
15     <xs:complexType>
16       <xs:sequence>
17         <xs:element maxOccurs="unbounded" ref="li"/>
18       </xs:sequence>
19     </xs:complexType>
20   </xs:element>
21   <xs:element name="li">
22     <xs:complexType>
23       <xs:sequence>
24         <xs:element ref="place"/>
25       </xs:sequence>
26     </xs:complexType>
27   </xs:element>
28   <xs:element name="place">
29     <xs:complexType>
30       <xs:simpleContent>
31         <xs:extension base="xs:NCName">
32           <xs:attribute name="qid" use="required" type="xs:NCName"/>
33         </xs:extension>
34       </xs:simpleContent>
35     </xs:complexType>
36   </xs:element>
37 </xs:schema>
```

Some explanations: lines 5–13 define the global element `statement`. Line 6 states that `statement` is of “complex type,” i.e., potentially containing other nested elements. Line 7 says that `statement` contains a sequence of elements `h1` and `ul`. The definition of `h1` on line 13 is extremely simple: it can only contain data of the type `xs:string` (as defined in [1, §3.2.1]). The element `ul` is defined in lines 14–20 as an element of complex type containing a sequence again. Still, this time the unique element appearing in the sequence, namely `li`, has a maximal quantifier: `unbounded`. Notice that line 17 does not define `li` but refers to its definition on lines 21–27. Element `li` is defined as a sequence of elements `place` (no quantifier, so it must occur exactly once), and `place` is defined on lines 28–36 as a complex type but with a simple content, meaning that it can have no nested elements but it does contain attributes. The textual contents of `place` and the value of the attribute `qid` are both of type `xs:NCName`, which, according to [1, §3.3.7], is an XML name without a colon (the "NC" in "NCName" stands for "Non Colonized").

The choice of `xs:NCName` for both the element `place` and the attribute `qid` is not optimal. The former should be broadened (a toponym can include whitespace, as in "New York"), and we will replace `xs:NCName` by `xs:string`. The latter can be more strict, as a Wikidata QID value is necessarily a "Q" followed by an integer. Here is the replacement code for the definition of `element`:

```
<xs:element name="place">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="qid" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="Q[0-9]+"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
 </xs:element>
```

We recognize in the `value` attribute of the `pattern` element the regular expression "Q followed by one or more Arabic digits."

13.1.5.1 Validation in Python

You can use the *xmlschema* package to validate an XML document vs. an XML schema. The code is straightforward:

```
import xmlschema
schema = xmlschema.XMLSchema('places.xsd')
print(schema.validate('places.xml'))
```

If the script returns `None`, one gets an error message pointing at the first discrepancy between the grammar and the document.

13.1.6 Parsing XML Documents

Once we know an XML document is well-formed (and potentially valid), we can *parse* it to extract information from it or transform it into another kind of document. There are at least two kinds of parsers:

1. those that parse the document's serialization: they will read the document, and whenever they detect a tag, they will act according to instructions provided to them. The prototypical parser of this category is *SAX* (Simple API for XML);
2. those that build an XML tree in memory and allow the user to play with it, move nodes around, create new nodes, serialize the tree, etc. The prototypical parser of this category is *DOM* (Document Object Model).

SAX and DOM and complementary: the former is useful when a document is large, and one only needs to have small parts in memory for a given task, or when the document is not very clean—SAX adopts a *local* view of your document. The latter is the "real thing": it stores the entire document in memory and gives unlimited possibilities, provided the document is well-formed and fits in memory—DOM adopts a *global* view of your document.

If XML tags mark named entities in a document, and one wants to extract them, then one should definitely use SAX. If, e.g., one wants to create a table of contents at the beginning of a document out of titles scattered throughout the document, then one should definitely use DOM, be it only for the reason that SAX reads the document linearly from beginning to end: if one wants to build the table of contents in SAX then one has to store it somewhere and add it to the document at a second run—and this would be reinventing DOM…

13.1.6.1 SAX

The *xml.sax* Python package has a `xml.sax.ContentHandler` class with a `parse()` method for parsing XML files. This class does nothing. One has to create a new class that inherits from it, and one has to redefine the various methods that correspond to tags detected in the document: `startElement` for opening element tags, `endElement` for closing element tags, `characters` for textual content (caveat: textual content is read in bunches of limited size so if one is looking for something specific, it may be broken between bunches), `startDocument` when at document begin, `endDocument` when parsing is finished, etc.

Here is a simple example where the desired output is, for each place, the place name followed by the Wikidata QID between parentheses:

```
import xml.sax

class CustomContentHandler(xml.sax.ContentHandler):
    def __init__(self):
        self.print = False

    def startElement(self, tagName, attrs):
```

```
        if tagName == 'place':
            self.store_qid=attrs['qid']
            self.print=True

    def endElement(self, tagName):
        if tagName == 'place':
            self.print = False

    def characters(self, chars):
        if self.print:
            print(chars + ' (' + self.store_qid + ')')

def main():
    handler = CustomContentHandler()
    xml.sax.parse('places.xml', handler)

if __name__ == '__main__':
    main()
```

We have chosen this example because it illustrates the difficulty of the SAX approach: we wish to display the QID code *after* the place name, but in the document, it arrives *before* it. The solution is to store the information in a variable. Similarly, we want to output *only* place names, so we need a boolean that becomes true only when the appropriate opening tag is detected and false when we leave the `place` element. In the code, one should normally check the existence of the `qid` attribute before writing `attrs['qid']` since the dictionary key may not exist. Still, the attribute's existence is guaranteed by the document's validity, which we have checked in Section 13.1.5.

13.1.6.2 XPath

XPath is a programming language for searching and accessing parts of the abstract syntax tree of an XML document. It is used in many tools, but mainly when parsing a document through DOM since it allows a precise definition of the information we wish to extract.

As the word "Path" in the name suggests, the syntax is inspired by one of the file paths in Unix. A slash `/` separates parent and child elements, similar to Unix paths' ☞ 162 directory separation. The metaphor XML DOCUMENT IS UNIX FILE STRUCTURE is fine on the syntax level. Still, there is a glitch: in Unix, there cannot be two directory or file siblings with the same name, while in XML, it is very common. After all, the list in our document is a sequence of arbitrary lengths of elements of the same name `li`, all included in the same element `ul`. A double slash `//`, as in URLs, denotes the document's root.

Therefore, XPath adds *tests* that are indices or expressions returning a boolean included in brackets after the parent directory name. For example, to access the value of the attribute of the second element `place` in our document, we could ask

```
//statement/ul/li[2]/place/@qid
```

the [2] meaning that we want the second li element and @ meaning that qid is an attribute. If we wanted the last li, we would write

```
//statement/ul/li[position()=last()]/place/@qid
```

Tests can be combined. For example, we could ask for the second li element only if the textual contents of place start with an 'S':

```
//statement/ul/li[2][fn:substring(place/text(),1,1)="S"]/place/@qid
```

where the first test is indexical: we take the second li under the node ul. The second test is boolean: we verify whether the substring of length 1, starting at the first character (the first character is position 1, not 0 as in Python) of the text node underneath place underneath li is an 'S.' If this test is true then we keep the li and go further down to access the qid attribute of the (unique) place element.

XPath version 3.1 [19, 13] is a very powerful language, and the example above only scratches its surface.

13.1.6.3 DOM

To illustrate *DOM* we will use the Python package *lxml*[2] which is compatible with XPath, while the official *xml.dom* is not. Here is the code to obtain the same output as in the SAX program of Section 13.1.6.1:

```
1  from lxml import etree
2  root = etree.parse("parse.xml").getroot()
3  places = root.xpath("//statement/ul/li/place")
4
5  for place in places:
6      print(place.text + " (" + place.get("qid") + ")")
```

On line 2, we parse the XML document, keep it in memory, and request the root element. On line 3, we apply an XPath path to the root element and call the list of obtained nodes, places. Then, on line 5, we iterate over places and output its text contents and the value of the qid argument for each element.

13.2 TEI

The *Text Encoding Initiative* (TEI) is an XML data format introduced in 1987 and continuously updated until today. Its goal is to encode any kind of text, including linguistic corpora, text in verse, performance text, transcriptions, dictionaries, bilingual texts, etc., and even non-textual resources such as trees, graphs, automata, and transducers.

A large number of corpora are encoded in TEI: the British National Corpus (100M word snapshot of current English), the Oxford Text Archive (1 GB of text in 25

[2] https://lxml.de/index.html

languages), the Perseus Project (Greek and Latin texts), Epidoc (epigraphy and papyrology), and also NLP-oriented corpora such as the Brown Corpus (one of the first corpora ever: 1M words in American English, compiled in 1967 [17]) and its British counterpart, the Lancaster-Oslo/Bergen Corpus, ELTeC (European Literary Text Collection), etc.

As a standard, TEI is gigantic. The documentation [24] (as of May 2023) is 2,033 pages long, in A4 paper size and 10-point text—and it contains an unlimited supply of funny quotes. There is not a single XML schema for TEI, but a Web site[3] where one can choose features necessary for a given corpus and obtain the corresponding grammar in any format (XML Schema, DTD, RELAX NG, etc.). The most global schema, called *tei_all*, contains the specification of no less than 2,943 elements.

TEI is divided into 21 modules [24, p. 2]. We will briefly present some of them in this section. The global element of a TEI document is `TEI` and the standard's namespace is `http://www.tei-c.org/ns/1.0`.

13.2.1 The Header

As many TEI documents are parts of our cultural heritage, it is important to identify them precisely and describe them as thoroughly as possible. The header is divided into many parts, the most important being:

1. a `fileDesc` element, a description of the data:
   a. a `titleStmt` element, the *title statement*, containing the title of the resource;
   b. a `publicationStmt` element: the *publication statement*, containing information on the distribution of the resource;
   c. a `sourceDesc` element: the *source description*, containing information not on the resource per se but on its source.

   The header can be recursive: if the resource is based, for example, on a book, then inside the `sourceDesc` we will find a new sequence of title statement and publication statement;
2. an `encodingDesc` element, information about the encoding of the file;
3. a `profileDesic` element, various non-bibliographic aspects of the text;
4. a `revisionDesc` element, the file's revision history.

We will illustrate the header part with the header of a resource based on a very important book: *The Complete Works of Shakespeare*, published in 1623, seven years after he died, in a 943-page volume.[4]

It starts with the title of the resource:

[3] `https://roma.tei-c.org/`

[4] Shakespeare, William, 1564-1616, *The Plays of William Shakespeare as published in the first folio of 1623*, Oxford Text Archive, `http://hdl.handle.net/20.500.12024/3014`.

```
<teiHeader>
  <fileDesc>
     <titleStmt>
        <title>The Plays of William Shakespeare as published
           in the first folio of 1623</title>
        <author>Shakespeare, William, 1564-1616</author>
     </titleStmt>
```

immediately followed by information on the distribution of the resource and, in particular, the permanent URL to download it:

```
<publicationStmt>
   <distributor>
      <name>University of Oxford Text Archive</name>
      <address>
         <addrLine>Oxford University Computing Services</addrLine>
         <addrLine>13 Banbury Road</addrLine>
         <addrLine>Oxford</addrLine>
         <addrLine>OX2 6NN</addrLine>
      </address>
   </distributor>
   <idno type="ota">https://hdl.handle.net/20.500.14106/3014</idno>
   <idno type="isbn10">1106000137</idno>
   <idno type="isbn13">9781106000132</idno>
   <availability status="free">
      <licence target="http://creativecommons.org/licenses/by-sa/3.0/">
      Distributed by the University of Oxford under a Creative Commons
      Attribution-ShareAlike 3.0 Unported License
      </licence>
   </availability>
</publicationStmt>
```

What follows is the description of the source of the contents of the resource:

```
<sourceDesc>
   <bibl>Revised version of  <relatedItem type="older"
      target="http://ota.ox.ac.uk/id/0119"/>
   </bibl>
   <bibl>
    The texts were originally prepared by Trevor
    Howard-Hill for use in his single volume concordances to
    Shakespeare (OUP, 1969f). They have since been reformatted
    to modern standards and carefully proofread by staff of
    Oxford University Press' Shakespeare Department for use in
    the new "Old Spelling" Oxford Shakespeare, under the
    general editorship of Dr Stanley Wells: <title>The complete works
    / William Shakespeare</title>; general editors, Stanley
    Wells and Gary Taylor ; editors Stanley Wells ... [et al.]
    ; with introductions by Stanley Wells. -- Oxford :
    Clarendon Press, 1986. -- (Oxford Shakespeare). -- ISBN
    0-19-812926-2
   </bibl>
```

And here we find out that the text originates from a facsimile of the first folio edition. The facsimile was published in 1902 by Sir Sidney Lee. It was based on a copy of the 1623 book belonging to the Duke of Devonshire:

```
<biblFull>
   <titleStmt>
      <title>Shakespeares comedies, histories &amp; tragedies : being a
      reproduction in facsimile of the first folio edition, 1623, from the
      Chatsworth copy in the possession of the Duke of Devonshire, K.G. /
      with introduction and census of copies by Sidney Lee</title>
      <author>Shakespeare, William, 1564-1616</author>
      <editor role="editor">Lee, Sidney, Sir, 1859-1926</editor>
   </titleStmt>
   <extent>xxxv, 908 p. : facsims. ; 39 cm.</extent>
   <publicationStmt>
      <publisher>Clarendon Press</publisher>
      <pubPlace>Oxford</pubPlace>
      <date>1902</date>
   </publicationStmt>
   <notesStmt>
```

The notes part, which we omit for the sake of brevity, contains metadata from the facsimile, including the table of contents.

```
         ...
         </notesStmt>
      </biblFull>
   </sourceDesc>
</fileDesc>
```

The next part is the encoding description. Its first sub-element is a description of ligatures represented by `<glyph/>` elements. The first one is a "$\overset{\text{u}}{\text{y}}$" ligature used in Elizabethan[5] typesetting, and meaning "thou":

```
<encodingDesc>
   <charDecl>
      <glyph xml:id="ythou">
         <glyphName>Lower case y with smaller lower case u above</glyphName>
         <desc>Abbreviation for thou in Elizabethan typesetting</desc>
         <charProp>
            <localName>entity</localName>
            <value>ythou</value>
         </charProp>
      </glyph>
      ...
   </charDecl>
```

Encoding description contains also bibliographic taxonomy:

```
   <classDecl>
      <taxonomy xml:id="OTASH">
         <bibl>University of Oxford Text Archive Subject Headings</bibl>
      </taxonomy>
      <taxonomy xml:id="LCSH">
         <bibl>Library of Congress Subject Headings</bibl>
      </taxonomy>
   </classDecl>
```

[5] Elizabeth I (1533–1603), is the queen that appears in many Doctor Who episodes, most importantly in the 2013 special episode *The Day of the Doctor*.

```
</encodingDesc>
```

Next comes the profile description, containing possible date of publication, language, and keywords in the LCSH (Library of Congress Subject Headings) scheme:

```
<profileDesc>
   <creation>
      <date notAfter="1623"/>
   </creation>
   <langUsage>
      <language ident="eng">English</language>
   </langUsage>
   <textClass>
      <keywords scheme="#LCSH">
         <term type="genre">Plays -- England -- 16th century</term>
         <term type="genre">Plays -- England -- 17th century</term>
         <term type="genre">Comedies -- England -- 16th century</term>
         <term type="genre">Comedies -- England -- 17th century</term>
         <term type="genre">Tragedies -- England -- 16th century</term>
         <term type="genre">Tragedies -- England -- 17th century</term>
      </keywords>
   </textClass>
</profileDesc>
```

And, last but not least, the revision description: the file has been modified only once, in 2010, when the header was updated to be conformant with the latest TEI specification:

```
  <revisionDesc>
     <change when="2010-08-31">Header normalised</change>
  </revisionDesc>
</teiHeader>
```

Needless to say the elements used in our example are only a tiny fraction of what is available in the TEI standard, cf. [24, Chapter 2].

13.2.2 The Common Core

This is a large part of the standard. It contains many submodules, the most important ones being:

- standard text elements [24, §3.1, 3.3]: `p` for paragraph, `emph` for emphasis, `foreign` for foreign words or phrases, `q` for quotes inside the text, `said` for passages thought or spoken aloud by real people of fictional characters, as in

```
<p>
<said who="#Clara" toWHom="#TheDoc">
I don't know why I asked that.
It's just, I like making
<foreign xml:lang="fr">soufflés</foreign>.
</said>
<said who="#TheDoc" toWhom="#Clara">
```

```
<emph><foreign xml:lang="fr">Soufflés</foreign>?</emph>
</said>
</p>
```

where the hashtags indicate that the keywords are references to persons described elsewhere in the document:

```
<standOff>
  <listPerson type="speakers">
    <person xml:id="TheDoc">
      <persName>The Doctor</persName>
    </person>
    <person xml:id="Clara">
      <persName>Clara Oswald</persName>
    </person>
  </listPerson>
</standOff>
```

- elements for editorial changes, additions, deletions, and omissions [24, §3.5];
- elements for named entities, numbers, dates, abbreviations, addresses [24, §3.6];
- lists [24, §3.8];
- annotations [24, §3.9], such as notes (not only footnotes and marginal notes, but any notes) and indexes;
- insertion of non-textual elements [24, §3.10] such as figures, media, binary data of any kind;
- bibliographic references [24, §3.12].

A text can be encoded in an element `text`, containing frontmatter `front`, the main body `body`, and backmatter `back`. In the body, one can have divisions `div` (or hierarchical divisions `div1`, `div2`, etc.), potentially containing heads `head`.

13.2.3 Linguistic Corpora

A corpus being, by definition, a collection of documents, TEI provides a `teiCorpus` element that can contain several documents enclosed in `TEI` elements. In the header of a corpus, one has additional descriptors [24, §15.2]:

- `textDesc` describes the text in terms of its situational parameters;
- `particDesc` describes the participants in any text;
- `settingDesc` describes the setting where a language interaction occurs.

Here is an example (taken from [24, p. 542]) of the text description of a corpus of conversations:

```
<textDesc n="Informal domestic conversation">
  <channel mode="s">informal face-to-face conversation</channel>
  <constitution type="single">each text represents a continuously
    recorded interaction among the specified participants
  </constitution>
  <derivation type="original"/>
```

```
    <domain type="domestic">plans for coming week, local affairs</domain>
    <factuality type="mixed">mostly factual, some jokes</factuality>
    <interaction type="complete" active="plural" passive="many"/>
    <preparedness type="spontaneous"/>
    <purpose type="entertain" degree="high"/>
    <purpose type="inform" degree="medium"/>
  </textDesc>
```

Here is a description of the participants to these conversations:

```
  <person sex="2" age="mid">
    <birth when="1950-01-12">
      <date>12 Jan 1950</date>
      <name type="place">Shropshire, UK</name>
    </birth>
    <langKnowledge tags="en fr">
      <langKnown level="first" tag="en">English</langKnown>
      <langKnown tag="fr">French</langKnown>
    </langKnowledge>
    <residence>Long term resident of Hull</residence>
    <education>University postgraduate</education>
    <occupation>Unknown</occupation>
    <socecStatus scheme="#pep" code="#b2"/>
  </person>
```

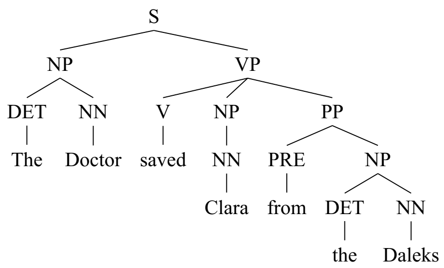
and finally the setting description:

```
  <settingDesc>
    <setting who="#p1 #p2">
      <name type="city">Bedford</name>
      <name type="region">UK: South East</name>
      <date>early spring, 1989</date>
      <locale>rug of a suburban home</locale>
      <activity>playing</activity>
    </setting>
  ...
  </settingDesc>
```

where `#p1` and `#p2` are persons involved in the setting.

13.2.4 Linguistic Annotation

As for linguistic annotation of texts, TEI provides elements for sentences `s`, phrases `phr`, and words `w`. By phrases are meant constituents, so one can describe a complete syntax tree by TEI tags. As an example, let us take the syntax tree from Section 5.3: ☞ 92



Here is how this tree can be described in TEI:

```
<s type="sentence">
  <phr ana="#S">
    <phr ana="#NP">
      <w ana="#DET">The</w>
      <w ana="#NN">Doctor</w>
    </phr>
    <phr ana="#VP">
      <w ana="#V" lemma="save">saved</w>
      <phr ana="#NP">
        <w ana="#NN">Clara</w>
      </phr>
      <phr ana="#PP">
        <w ana="#PRE">from</w>
        <phr ana="#NP">
          <w ana="#DET">the</w>
          <w ana="#NN" lemma="Dalek">Daleks</w>
        </phr>
      </phr>
    </phr>
  </phr>
</s>
```

where the hashtags once again denote keywords that have been defined elsewhere in the document in an "interpretation group," such as:

```
<interpGrp type="POS">
    <interp xml:id="S">Sentence</interp>
    <interp xml:id="NP">Noun phrase</interp>
    ...
</interpGrp>
```

☞ 351 There is also a `norm` attribute for normalized forms, as in *Flowers for Algernon*:

```
<w>where</w> <w>I</w> <w>go</w> <w>to</w> <w norm="learn">lern</w>
<w norm="reading">reeding</w>...
```

There exists a more heavy-duty representation of linguistic features in TEI. This module has become an ISO standard [10]. We will not cover it here, but the interested reader can find more information in [24, §18].

13.2.5 Transcriptions of Speech

A speech transcription needs a particular element `recordingStmt` [24, §8.2] containing information about the recording. As we have seen previously, speakers are identified in a `listPerson` list.

The basic element for transcribing speech is `u`, for *utterance*. Whether inside or between utterances, we can have pauses `pause`, incidents `incident`, vocals `vocal`, and gestures `kinesic`. When speech quality changes, one can use the `shift` element. When written text appears in the movie or broadcast being transcribed, one can use the `writing` element. Here is an example, from the episode *The Pandorica Opens* S31E12:

```
<u who="#Amy">There it is. You remember.
  This is you, and you are staying.</u>
<incident>
  <desc>Rory's gun hand activates.</desc>
</incident>
<u who="#Rory">No.
  <kinesic>
    <desc>Rory shoots Amy.</desc>
  </kinesic>
<shift new="shouting"/>No! No! No!</u>
```

When necessary, one can synchronize utterances with a timeline. One first defines time instances:

```
<timeline origin="#TS-t01" unit="s">
  <when xml:id="TS-t01" absolute="15:33:01Z"/>
  <when xml:id="TS-t02" interval="2.5" since="#TS-t01"/>
  ...
</timeline>
```

and then one can use the identifiers in `anchor` elements:

```
<u who="#Rory"><anchor synch="#TS-t01/>No! No!
  No!<anchor synch="#TS-t02/></u>
```

There is even the possibility of adding prosodic information [24, §8.4.4] and speech management information [24, §8.4.5], for example, if words are unclear because of some event, but the transcriber wishes to include her hypothesis, she may write:

```
<unclear reason="exploding-tardis">Geronimo!</unclear>
```

13.2.6 Verses and Alignment

> What makes Goethe's poem hopelessly difficult to translate is, paradoxically, its very lack of difficulty
>
> Walter Silz [22, p. 345]

Verse lines are marked by `l` elements. Stanzas are marked by `lg` (line group) elements.

To align texts, one can use unique identifiers and anchors. Here is an example: we will align the original text of Goethe's *Über allen Gipfeln*, verses that he wrote on the board wall of a hunting lodge on the Kickelhahn (Ilmenau, Central Thuringian Forest, Germany) on September 6th, 1780 [18], with the Greek translation by Georgios Vizyinos, one of the greatest Greek writers. His translation of the poem is included in the novel *Αἱ συνέπειαι τῆς παλαιᾶς ἱστορίας* ⌊The Consequences of the Old Story⌋ (1884), the plot of which is set in Clausthal, in the Harz mountains. We will compare these two versions with the English translation by Henry Wadsworth Longfellow (1844) [22]. The three versions have eight verses, but their alignment is not entirely trivial:

```
<div xml:id="goethe" xml:lang="de">
  <lg>
    <l><seg xml:id="d1">Über allen Gipfeln</seg></l>
    <l><seg xml:id="d2">Ist Ruh',</seg></l>
    <l><seg xml:id="d3">In allen Wipfeln</seg></l>
    <l><seg xml:id="d4">Spürest Du</seg></l>
    <l><seg xml:id="d5">Kaum einen Hauch;</seg></l>
    <l><seg xml:id="d6a">Die Vögelein</seg>
       <seg xml:id="d6b">schweigen</seg>
       <seg id:xml="d6c">im Walde.</seg></l>
    <l><seg xml:id="d7">Warte nur! Balde</seg></l>
    <l><seg xml:id="d8">Ruhest du auch.</seg></l>
  </lg>
</div>
<div xml:id="longfellow" xml:lang="en">
  <lg>
    <l><seg xml:id="e1">O'er all the hill-tops</seg></l>
    <l><seg xml:id="e2">Is quiet now,</seg></l>
    <l><seg xml:id="e3">In all the tree-tops</seg></l>
    <l><seg xml:id="e4">Hearest thou</seg></l>
    <l><seg xml:id="e5">Hardly a breath;</seg></l>
    <l><seg xml:id="e6a">The birds</seg>
       <seg xml:id="e6d">are asleep</seg>
       <seg id:xml="e6c">in the trees:</seg></l>
    <l><seg xml:id="e7">Wait; soon like these</seg></l>
    <l><seg xml:id="e8">Thou too shalt rest.</seg></l>
  </lg>
</div>
<div xml:id="vizyinos" xml:lang="el">
  <lg>
    <l><seg xml:id="g1">Ἐπὶ πάντων τῶν ὀρέων</seg></l>
```

Über allen Gipfeln	O'er all the hill-tops	Ἐπὶ πάντων τῶν ὀρέων
Ist Ruh',	Is quiet now,	ἡσυχία βασιλεύει.
In allen Wipfeln	In all the tree-tops	Ἐπὶ τῶν κλαδίσκων πλέον
Spürest Du	Hearest thou	οὔτε φύλλον δὲν σαλεύει.
Kaum einen Hauch;	Hardly a breath;	Τὰ πτηνὰ ταῖρι ταῖρι
Die Vögelein schweigen im Walde.	The birds are asleep in the trees:	κοιμῶνται σιγὰ κ᾽ εὐτυχῆ.
Warte nur! Balde	Wait; soon like these	Ὦ, καρτέρει, καρτέρει,
Ruhest du auch.	Thou too shalt rest.	καὶ σὺ θὰ κοιμᾶσ᾽ ἐν βραχεῖ!

Fig. 13.2 The original text and two translations of Goethe's *Über allen Gipfeln* with aligned segments marked by background colors

```
      <l><seg xml:id="g2">ἡσυχία βασιλεύει.</seg></l>
      <l><seg xml:id="g3">Ἐπὶ τῶν κλαδίσκων πλέον</seg></l>
      <l>οὔτε φύλλον δὲν σαλεύει.</l>
      <l><seg xml:id="g6a">Τὰ πτηνὰ</seg> ταῖρι ταῖρι</l>
      <l><seg xml:id="g6d">κοιμῶνται</seg> <seg xml:id="g6b">σιγὰ</seg>
         κ᾽ εὐτυχῆ.</l>
      <l><seg xml:id="g7a">Ὦ, καρτέρει, καρτέρει,</seg></l>
      <l><seg xml:id="g8">καὶ σὺ θὰ κοιμᾶσ᾽</seg>
         <seg xml:id="g7b">ἐν βραχεῖ!</seg></l>
    </lg>
  </body>
```

As shown in Fig. 13.2, the three first verses are perfectly aligned. The 3rd and 4th verses are similar in German and English (even though Goethe refers to feeling and Longfellow to hearing), while the Greek 4th verse translates the feeling of quietness by an image: "No leave is trembling." "Birds" appear in the German 6th verse and the English 6th verse but a bit earlier in the 5th verse for Greek. Vizyinos fills the remaining part of the fifth verse with "(the birds sleep) in pairs," increasing the coziness of the mood. Interestingly, for Goethe, the birds "are silent," while for Longfellow, they "are asleep," and Vizyinos uses both: "(the birds) are sleeping silently." Also, Goethe and Longfellow have their birds sleeping in the forest or trees, while Vizyinos qualifies them as sleeping happily without indication of location. The 7th and the 8th verses are well aligned, except that the word "soon" appears at the end of the 8th verse for the Greek text.

We have created the corresponding segments and marked them with the `seg` tag and a unique identifier for each. The next step is to use the `linkGrp` element to provide a sequence of links between segments in the three texts:

```
<linkGrp type="alignment" domains="#goethe #longfellow #vizyinos">
  <link target="#d1 #e1 #g1"/>
  <link target="#d2 #e2 #g2"/>
  <link target="#d3 #e3 #g3"/>
  <link target="#d4 #e4"/>          <!-- hearest thou -->
  <link target="#d5 #e5"/>          <!-- hardly a breath -->
  <link target="#d6a #e6a #g6a"/>   <!-- the birds -->
  <link target="#d6b #g6b"/>        <!-- are silent -->
  <link target="#d6c #e6c"/>        <!-- in the wood/trees -->
  <link target="#e6d #g6d"/>        <!-- are asleep -->
  <link target="#d7 #e7 #g7a #g7b"/> <!-- wait; soon like these -->
  <link target="#d8 #e8 #g8"/>      <!-- thou too shalt rest -->
```

```
</linkGrp>
```

The attribute `domains` identifies the text units in which the segments are located. This method allows a precise alignment, but there is a hitch: segments cannot overlap, and they have to be nested into their parent structures, in this case, `l` elements for verses. It is for this reason that we have created two segments `g7a` and `g7b` so that their combination can be aligned with `d7` and with `e7`. This problem comes from the fact that by using XML, we have chosen tree structures to represent our data and have to live with this choice.

The solution is to remove the `seg` markup, to add unique identifiers to the verses, and to use the `#string-range` function [24, §16.2.4.7] instead of unique identifiers. This will allow us to identify the (possibly not contiguous) character strings of the segments we want to identify. To illustrate this technique, let us take the first and the two last verses of the Greek text:

```
<div xml:id="vizyinos" xml:lang="el">
  <lg>
    <l><seg xml:id="g1">Ἐπὶ πάντων τῶν ὀρέων</seg></l>
    ...
    <l><seg xml:id="g7a">Ὦ, καρτέρει, καρτέρει,</seg></l>
    <l><seg xml:id="g8">καὶ σὺ θὰ κοιμᾶσ᾽</seg>
       <seg xml:id="g7b">ἐν βραχεῖ!</seg></l>
  </lg>
</body>
```

and rewrite them as:

```
<div xml:id="vizyinos" xml:lang="el">
  <lg xml:id="g" xml:space="preserve">
    <l xml:id="g1">Ἐπὶ πάντων τῶν ὀρέων</l>

    ...
    <l xml:id="g7">Ὦ, καρτέρει, καρτέρει,</l>
    <l xml:id="g8">καὶ σὺ θὰ κοιμᾶσ᾽ ἐν βραχεῖ!</l>
  </lg>
</body>
```

Now, we have uniquely identified the verse group and the individual verses. To refer to the textual contents of the first verse, we can write

```
#string-range(//l[@xml:id='g1'],0,20)
```

meaning: find the node of type `l` uniquely identified by `g1`, and fetch the string starting at position 0 and ending at position 20. That is exactly "Ἐπὶ πάντων τῶν ὀρέων"EL. The difficulty is to merge verse 7 and the last two words of verse 8 into a
☞ 368 single string. For that, the XPath-like path in the first argument of `#string-range` has to stop at `lg` and then we have to count the distance of "Ὦ, καρτέρει, καρτέρει,"EL from the begin of the textual contents of `lg`.

This distance is calculated by removing all markup and keeping only the text leaves. As we used the attribute `xml:space="preserved"`, whitespace indeed counts in our calculations. We found that letter "Ὦ"EL is at offset 163 of `lg`'s textual content. This verse is 23 characters long, so we arrive at position 186, then we skip to offset 208 to incorporate “ἐν βραχεῖ!”$_{\mathrm{EL}}$ with an initial space, and we finish at position 219. The link to obtain the merge of the two character strings

```
Ὦ, καρτέρει, καρτέρει, ἐν βραχεῖ!
```

is therefore

```
#string-range(//lg[@xml:id='g'],163,186,208,219)
```

The function `#string-range` can gather an arbitrary number of character strings, as long as we add their offsets as additional argument pairs to it.

13.2.7 Dictionaries

The structure of dictionary data is covered by ISO Standard 24613 [20, 11], also known as LMF (Lexical Markup Framework). LMF defines (abstract) classes and attributes in the native UML style. The fourth part of the standard ([12]) describes the “serialization” of LMF in TEI markup.

A dictionary contains `entry` elements [24, §9.2], containing one or more `form`s, followed by one or more `sense`s. A form has an orthographic representation (`orth`) and a phonetic representation (`pron`). Both can be specialized to specific regions. We can have an element `etym` with etymological information between forms and senses. For each sense, we can have grammatical information (`gramGrp`), definitions (`def`), examples (`cit`), cross-references to other entries (`xr`) using pointers (`ptr`), and related entries, which are sub-entries that are expanded because they are not important enough to become independent entries, etc.

We will illustrate dictionary encoding by an excerpt from the *Hacker's Dictionary* by Steele [23], the (adapted) entry of the word “ping”:

```
<entry xml:id="ping">
  <form>
    <orth>ping</orth>
    <pron>pɪŋ</pron>
  </form>
  <etym>From the submariners' term for a sonar pulse</etym>
  <sense n="1">
    <gramGrp><pos>noun</pos></gramGrp>
    <def>Slang term for a small network message
      (<mentioned>ICMP ECHO</mentioned>) sent by a computer
      to check for the presence and alertness of another.
      Occasionally used as a phone greeting.</def>
    <xr type="see">See <ref target="#ACK"/>,
      also <ref target="#ENQ"/></xr>
  </sense>
  <sense n="2">
    <gramGrp><pos>verb</pos></gramGrp>
    <def>To verify the presence of</def>
  </sense>
  <sense n="3">
    <gramGrp><pos>verb</pos></gramGrp>
```

```
      <def>To get the attention of</def>
    </sense>
    <sense n="4">
      <gramGrp><pos>verb</pos></gramGrp>
      <def>To send a message to all members of a
        <ref target="#mailing_list"/> requesting an
        <ref target="#ACK"/> (in order to verify that
        everybody's addresses are reachable).</def>
      <cit type="example">We haven't heard much of anything from Geoff,
        but he did respond with an ACK both times I pinged jargon-friends.
      </cit>
    </sense>
```

After four senses that reflect more the role of the Unix utility *ping*, here is a more metaphoric fifth sense, with a derived noun: *pingfulness*, the state of people who are so happy that they literally exude pings, and an antonym: *blargh*, a state of frustration.

```
    <sense n="5">
      <gramGrp><pos>noun</pos></gramGrp>
      <def>A quantum packet of happiness. People who are very happy
        tend to exude pings; furthermore, one can intentionally create
        pings and aim them at a needy party (e.g., a depressed person).
        This sense of ping may appear as an exclamation; "Ping!"</def>
      <cit type="example">I'm happy; I am emitting a quantum of happiness;
        I have been struck by a quantum of happiness.</cit>
      <re type="derived">
        <form>
          <orth>pingfulness</orth>
          <pron>'pɪŋfᵊlnəs</pron>
          <hyph>ping-ful-ness</hyph>
        </form>
        <sense>
          <def>The state of people who exude pings</def>
        </sense>
      </re>
      <xr type="antonym">Oppose <ref target="#blargh"/></xr>
    </sense>
  </entry>
```

13.3 CDL

☞ 55 An analytic approach to sinograms, that is an application for building sinograms out of components, components out of smaller parts, and smaller parts out of the
☞ 55 39 fundamental strokes of Chinese calligraphy [7, pp. 154–155], may sound like a heresy to calligraphers. It becomes a necessity for tasks such as font creation. After all, an average Japanese or Chinese font contains at least 20k glyphs. Furthermore, the repertoire of sinograms is not fixed. Every year, new sinograms are invented and proposed to Unicode for insertion, not to mention historical characters that appear in old documents and must be encoded.

There are also cases where new sinograms were invented for specific reasons: to connote the low IQ of the author, as in *Flowers for Algernon* (cf. §12.9 and [9, p. 278]), or to translate Lewis Carroll's neologisms in the *Jabberwocky* poem [15], or the 4,000 invented sinograms of Xu Bing's *Book from the Sky* (Princeton University Art Museum) [21] (see also p. 188).

☞ 351

☞ 50

There have been several attempts to describe sinograms analytically [8], be it by generative grammar [25], the Prolog programming language [5], METAPOST [26], etc. The only one that covers the newest Unicode versions and describes *all* Unicode characters is CDL.

CDL (Character Description Language) [2, 3] is an XML data format for describing sinograms developed by Tom Bishop and Richard Cook of the Wenlin Institute.

Compared to TEI, CDL is extremely simple: it manages to describe more than 100k sinograms with only four XML elements: `cdl-list` as the global element of a document containing sinogram descriptions, `cdl` for a description, `comp` for a component used in a description, `stroke` for one of the 39 fundamental strokes. The subtlety of CDL lies in the fact that it is both logically structured (a sinogram can be described by components having nested components, and so on, up to the stroke level) and geometric (bounding boxes for each component and each stroke, in a 128 × 128 grid).

Here is an example. The very frequent sinogram “行”JA consists of two horizontally-aligned components “彳”JA and “亍”JA—its CDL description is simply

```
<cdl char="行" uni="884C">
  <comp char="彳" uni="5F73" points="0,0 40,128"/>
  <comp char="亍" uni="4E8D" points="60,12 128,128"/>
</cdl>
```

Here, the bounding box is simply a box of full height and a width of 40 units (among 128) for the left component and a box of full height and width of 68 units for the right component (20 units are left to separate the two).

Here is how the component “彳”JA is described:

```
<cdl char="彳" uni="5F73">
  <stroke type="p" points="107,0 10,46"/>
  <stroke type="p" points="128,38 0,83"/>
  <stroke type="s" points="86,70 86,128"/>
</cdl>
```

The stroke abbreviations stand for *piě* (a diagonal stroke descending from right to left, it is the 5th fundamental stroke) and *shù* (a vertical stroke, the 3rd fundamental stroke).

The two methods can be combined. To obtain, for example, “太”JA, one can write:

```
<cdl char="太" uni="592A">
  <comp char="大" uni="5927" points="0,0 128,128"/>
  <stroke type="d" points="45,104 67,128"/>
</cdl>
```

where `d` stands for *diǎn* (a dot or very short segment, the 8th fundamental stroke).

Besides the bounding box, the CDL format also allows specifying the stroke order, the head (normal, cut, corner, or vertical), and the tail (normal, cut, or long) of each stroke.

As an exercise, let us build Algernon’s pseudo-sinogram (cf. §12.9) in DCL. The code will be

```
<cdl char='' uni='E000'>
  <comp char='彳' uni='5F73' points='0,0 32,128' />
  <comp char='売' uni='58F2' points='40,0 128,128' />
</cdl>
```

A screen capture of the *Wenlin* application sinogram construction window can be seen in Fig. 13.3.

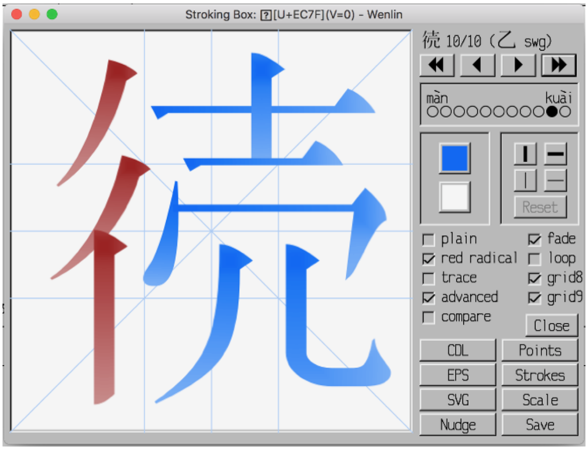


Fig. 13.3 Creating a glyph for the non-existing sinogram “” through *Wenlin*. Used by permission of Wenlin Institute

13.4 Exercises

Exercise 12-1: Jerome K. Jerome’s *Three Men on the Bummel* in TEI

Jerome K. Jerome (1859–1927) was an English author, best known for the comic book *Three Men in a Boat (to say nothing of the dog)* (1889), which has had an astonishing science-fiction sequel in 1997: *To Say Nothing of the Dog* by Connie Willis. Here is a text extracted from Jerome K. Jerome’s second most important book, *Three Men on the Bummel* (1900):

> The Kneipe is what we should call a stag party, and can be very harmless or very rowdy, according to its composition. One man invites his fellow-students, a dozen or a hundred, to a café, and provides them with as much beer and as many cheap cigars as their own sense of health and comfort may dictate, or the host may be the Korps itself. Here, as everywhere, you observe the German sense of discipline and order. As each new comer enters all those sitting round the table rise, and with heels close together salute. When the table is complete, a chairman is chosen, whose duty it is to give out the number of the songs. Printed books

> of these songs, one to each two men, lie round the table. The chairman gives out number twenty-nine. "First verse," he cries, and away all go, each two men holding a book between them exactly as two people might hold a hymn-book in church. There is a pause at the end of each verse until the chairman starts the company on the next. As every German is a trained singer, and as most of them have fair voices, the general effect is striking.
>
> Although the manner may be suggestive of the singing of hymns in church, the words of the songs are occasionally such as to correct this impression. But whether it be a patriotic song, a sentimental ballad, or a ditty of a nature that would shock the average young Englishman, all are sung through with stern earnestness, without a laugh, without a false note. At the end, the chairman calls "Prosit!" Everyone answers "Prosit!" and the next moment every glass is empty. The pianist rises and bows, and is bowed to in return; and then the Fräulein enters to refill the glasses.
>
> Between the songs, toasts are proposed and responded to; but there is little cheering, and less laughter. Smiles and grave nods of approval are considered as more seeming among German students.
>
> A particular toast, called a Salamander, accorded to some guest as a special distinction, is drunk with exceptional solemnity.
>
> "We will now," says the chairman, "a Salamander rub" ("Einen Salamander reiben"). We all rise, and stand like a regiment at attention.
>
> "Is the stuff prepared?" ("Sind die stoffe parat?") demands the chairman.
>
> "Sunt," we answer, with one voice.
>
> "Ad exercitium Salamandri," says the chairman, and we are ready.
>
> "Eins!" We rub our glasses with a circular motion on the table.
>
> "Zwei!" Again the glasses growl; also at "Drei!"
>
> "Drink!" ("Bibite!")
>
> And with mechanical unison every glass is emptied and held on high.
>
> "Eins!" says the chairman. The foot of every empty glass twirls upon the table, producing a sound as of the dragging back of a stony beach by a receding wave.
>
> "Zwei!" The roll swells and sinks again.
>
> "Drei!" The glasses strike the table with a single crash, and we are in our seats again.

The original 1900 edition of *Three Men on the Bummel* can be found on the Internet Archive[6].

Encode this text in TEI and validate it using the online TEI validation service.[7]

Exercise 12-2: Find the subject occurrences in Bach's Fugue BWV 846

It is generally accepted that linguistic data is semi-structured in the sense that there is a structure, but this structure is not explicit and must be extracted. Another kind of data can be qualified as being semi-structured: *music*. In this exercise, we will discover a Web standard for music, *MusicXML*[8] and we will use regular expressions to extract the main structural element of a popular compositional form in Western music: the *fugue*. In a fugue, we have 2, 3, and up to 6 "voices" that interact. J.S.

[6] https://archive.org/details/threemenonbummel00jerouoft/page/296/mode/2up

[7] https://teibyexample.org/exist/tools/TBEvalidator.htm

[8] https://www.musicxml.com

Bach's fugues are "abstract" music that can be played on a piano, a harpsichord, or an organ, but also on a violin or a cello, by a string quartet, or even by a jazz trio. We have chosen one of the most beautiful fugues ever written, J.S. Bach's fugue in C, from the *Well-Tempered Clavier*, identified as BWV 846. This fugue is astonishingly dense: the subject occurs 24 times, in only 27 measures—as Keller says, "one entry follows upon the heels of another" [14, p. 51]. He also notices that the subject consists of fourteen notes, and fourteen is the sum of the letters of the name BACH, so

> Bach may have hidden his name in the structure of his work in much the same fashion as a master architect might have done in the Middle Ages. [14, p. 50]

Your task is to find as many occurrences of the subject as possible. Explain why some occurrences cannot be found algorithmically, or at least not as easily.

Exercise 12-3: Document conversion from HTML to TEI using SAX and DOM

Convert the following HTML document into TEI using SAX and DOM:

```
<html>
<body>
<h1 xml:id="thraupidae">Thraupidae</h1>
<p>Tanagers, 240 species of mostly brightly colored fruit-eating birds.</p>
    <h2 xml:id="poospiza">Poospiza</h2>
    <p>Arboreal feeders in light woodland and scrub.</p>
        <h3 xml:id="poospiza-h">Poospiza hypochondria</h3>
        <p>A rufous-sized warbling finch.</p>
        <h3 xml:id="poospiza-a">Poospiza avenseguim</h3>
        <p>A nonsense name inspired by Harry Potter.</p>
    <h2 xml:id="saltatricula">Saltatricula</h2>
    <p>A genus of South-American seed-eating birds.</p>
        <h3 xml:id="saltratricula-m">Saltratricula multicolor</h3>
        <p>A many-colored chaco-finch.</p>
<h1 xml:id="thamnophilidae">Thamnophilidae</h1>
<p>Antbirds are generally small birds with rounded wings and strong
    legs.</p>
    <h2 xml:id="rhopornis">Rhopornis</h2>
    <p>The slender antbird is endemic to dry forests in Bahia and Minas
        Gerais in Brazil.</p>
</body>
</html>
```

Add a table of contents with links to the subdivision titles. Validate your result using the online TEI validation service.[9]

Compare the two approaches.

[9] https://teibyexample.org/exist/tools/TBEvalidator.htm

References

[1] Paul V. Biron and Ashok Malhotra. “XML Schema Part 2: Datatypes Second Edition”. https://www.w3.org/TR/xmlschema-2/. 2004.

[2] Tom Bishop. “A character description language for CJK”. In: *Multilingual* 91 (2007), pp. 62–68.

[3] Tom Bishop. “A Specification for CDL Character Description Language”. In: *Proceedings of the Glyph and Typesetting Workshop, Kyoto, 2003*. http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/098-cdl.pdf. 2003, pp. 98–103.

[4] Tim Bray et al. “Extensible Markup Language (XML) 1.0 (Fifth Edition)”. https://www.w3.org/TR/xml/. 2008.

[5] Martin J. Dürst. “Coordinate-independent font description using Kanji as an example”. In: *Electronic Publishing* 6.3 (1993), pp. 133–143.

[6] Julius J. Epstein et al. “Casablanca”. http://www.vincasa.com/casabla.pdf. 1942.

[7] Yannis Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O’Reilly, 2007.

[8] Yannis Haralambous. “New Perspectives in Sinographic Language Processing Through the Use of Character Structure”. In: *CICLing 2013: 14th International Conference on Intelligent Text Processing and Computational Linguistics, Samos*. Vol. 7816. Springer LNCS. 2013, pp. 201–217.

[9] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. “Graphemic and Graphetic Methods in Speculative Fiction”. In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 4. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 259–359.

[10] International Organization for Standardization. *ISO 24610-1:2006 Language resource management — Feature structures — Part 1: Feature structure representation*. 2006.

[11] International Organization for Standardization. *ISO 24613-1:2019 Language resource management — Lexical markup framework (LMF) — Part 1: Core model*. 2019.

[12] International Organization for Standardization. *ISO 24613-4:2021 Language resource management — Lexical markup framework (LMF) — Part 4: TEI serialization*. 2021.

[13] Michael Kay. “XPath and XQuery Functions and Operators 3.1”. https://www.w3.org/TR/xpath-functions-31. 2017.

[14] Hermann Keller. *The Well-Tempered Clavier by Johann Sebastian Bach*. W. W. Norton & Company, 1976.

[15] Alan Levinovitz. “Slaying the Chinese Jabberwock. The limitations of translating nonsense”. In: *The Believer* 78 (2011).

[16] Jonathan Marsh, Daniel Veillard, and Norman Walsh. “xml:id Version 1.0”. http://www.w3.org/TR/xml-id/. 2005.

[17] Francis W. Nelson and Henry Kucera. *Computational Analysis of Present-Day American English*. Providence, RI: Brown University Press, 1967.

[18] Elke Richter. “„Über allen Gipfeln ist Ruh…“ – Goethes Gedicht wird 240 Jahre alt”. Blog-Archiv, Klassik Stiftung Weimar https://perma.cc/55J8-SU7C. 2020.

[19] Jonathan Robie, Michael Dyck, and Josh Spiegel. “XML Path Language (XPath) 3.1”. https://www.w3.org/TR/xpath-31/. 2017.

[20] Laurent Romary et al. “LMF Reloaded”. arXiv:1906.02136. 2019.

[21] Jerome Silbergeld and Dora C. Y. Ching, eds. *Persistence/Transformation: Text as Image in the Art of Xu Bing*. Publications of the Tang Center for East Asian Art. Princeton University Press, 2006.

[22] Walter Silz. “Longfellow’s Translation of Goethe’s “Ueber allen Gipfeln…” ” In: *Modern Language Notes* 71 (1956), pp. 344–345.

[23] Guy L. Steele. *Hacker’s Dictionary: Guide to the World of Computer Wizards*. Harper & Row, 1996.

[24] The TEI Consortium. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html. 2023.

[25] Jason Chia-Seng Wang. “Toward a generative grammar of Chinese character structure and stroke order”. PhD thesis. University of Wisconsin-Madison, 1983.

[26] Candy L. K. Yiu and Wai Wong. “Chinese character synthesis using METAPOST”. In: *TUGboat* 24 (2003), pp. 85–93.

Part IV
Statistical Methods

Even before the computer, as we will see in this part, people were counting words, for various purposes. And like in Kubrick's *2001: A Space Odyssey*, the bone thrown by the ape becomes a space station, here also we have a technological miracle: word counting has resulted in ChatGPT. In this part we will briefly look at some intermediate steps of this evolution. Before deep learning, ordinary machine learning algorithms were used for NLP. At that time the most exciting task was data preprocessing, to achieve a form that could be handled by the machine learning algorithm. These methods were not very performant, compared to deep models, but were (at least partly) transparent. After that we entered the era of deep learning and nothing was ever the same again. Text is embedded into high-dimensional spaces with miraculous properties and models will read terabytes of data and endlessly go back and fort across millions of elementary cells in order to get trained. The results are known. ChatGPT appeared on *Time Magazine*'s cover on February 20, 2023. We asked it whether it is the technological singularity that Hawking has predicted, and which will result in the end of humanity. The reader can check its answer on p. 452.

Chapter 14
Counting Words

As we have seen in Chapters 2 to 7, language has a fabulous, partly-visible, and partly-hidden structure, and still a lot remains to be discovered. Up to now, our study has been qualitative: in second articulation, when meaning emerges out of phonemes or graphemes, meaning is a quality of the morpheme. Part-of-speech tags, belonging to a constituent, etc., are part of the qualitative information we can gather on morphemes, words, or word groups, and we obtain it by doing tests and answering questions.

A different approach is the *quantitative* one, i.e., to conclude by attaching numerical information to language units. This approach is not specific to the computer: For centuries, Jewish scribe-scholars called Masoretes, i.e., "masters of the tradition," have counted verses, words, and letters of the Bible. In the oldest complete manuscript of the Hebrew Bible, dating 1008 CE, the *Leningrad Codex*, at the end of the book of Deuteronomy, there is a note giving the sum of the verses (5,845), the sum of the words (79,856) and the sum of the letters (400,945) of the first five books, called the *Torah*. Masoretes were also searching the "mathematical centers" of the various books, metaphorically connecting a location in the text obtained by counting (verses, words, or letters) to the "central meaning" of the text [14, pp. 7–9]. Closer to us, in 1913, a Russian mathematician called A. A. Markov used Pushkin's novel *Eugene Onegin* as his first example of a method nowadays called "Markov chains": he took an excerpt of 20,000 graphemes from the novel, excluding the Cyrillic hard and soft signs ъ and ь (which were much more frequent before Lenin's reform of the language) and examined the patterns of vowel and consonant combinations [18]. In the thirties and forties of the twentieth century, counting words and other morphological or syntactic text features was a standard method for identifying authorship of texts [20, 7, 1, 3].

Well understood, computers are very good at counting things, and this is why statistical approaches to language have had a tremendous boost in the early days of computing and a second major boost in the nineties when the personal computer and the increase in processing power made elaborate statistical methods accessible to everyone.

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_14

14.1 Tokenization and Segmentation

Contrarily to the cold crispness and rigidity of logic, statistics have a sense of the *à peu près*, i.e., the approximation for practical purposes. As already mentioned, the ☞ 65 notion of "word" has never been adequately defined in linguistics. In other words, we lack words and proper wording to define what a word is (pun!). This is hardly a problem when we take the statistical approach: the problem is dodged by dedicating a step of our workflow to segment linguistic data into *tokens*, a particular case of tokens being word-like tokens. It is then an algorithm that decides how data are broken into "words" (in the following, we will refrain from using double quotes for simplicity).

Calling this step "tokenization" is quite clever since it frees us from preconceived ideas on what a "word" is or should be and makes this step language-dependent and ☞ 45 task-dependent. Tokens in some languages will be graphemic sequences separated by whitespace. There will be free morphemes or morpheme compounds in other languages, such as Thai or Chinese. As computers read text linearly, tokenization is equivalent to detecting boundaries between tokens, which is called *segmentation*.

Segmentation is crucial for NLP and very simple human-machine interaction: in many text editors or word processors, when we double-click on a "word," we expect it to be highlighted and selected for copying, searching, etc. The Unicode Standard Annex on segmentation [6] provides guidelines for this operation and, therefore, for developing segmentation and tokenization algorithms.

From the point of view of implementation, the simplest possible tokenization can be obtained by using regular expressions, particularly the Python `split()` method. Take, for example, the sentence

> Just in case you're wondering, all this I'm-messing-with-your-mind chat, I've heard worse on a Friday night in Sheffield.

from the Whovian corpus (S39E9). The `split()` method will return the tokens

```
[""Just","in","case","you're","wondering,","all","this",
"I'm-messing-with-your-mind","chat,","I've","heard","worse",
"on","a","Friday","night","in","Sheffield."]
```

where `in` occurs twice. There are at least three tokens that are dubious: `I'm-messing-with-your-mind`, `I've`, and `you're`, which are contractions, typical of speech. The first one carries a multitude of hyphens, hence converting a relatively long phrase into a single noun. The other problem is the presence of punctuation (commas and periods). If the task is to count words or gather them into classes, punctuation may be unnecessary. Still, one must be sure that a point denotes an end-of-sentence period and not an abbreviation mark, in which case it is necessary to keep it.

The *nltk* package provides a module dedicated to this task: `tokenize`. The program

```
from nltk.tokenize import word_tokenize
print(word_tokenize('''Just in case you're wondering, all this
I'm-messing-with-your-mind chat, I've heard worse on a Friday
night in Sheffield.'''))
```

returns

```
['Just', 'in', 'case', 'you', "'re", 'wondering', ',', 'all', 'this',
"I'm-messing-with-your-mind", 'chat', ',', 'I', "'ve", 'heard',
'worse', 'on', 'a', 'Friday', 'night', 'in', 'Sheffield', '.']
```

which, interestingly, separates `I` from `'ve` and `you` from `'re`, but keeps `I'm-messing-with-your-mind` in one piece. It also considers punctuation marks as individual tokens.

Using a package such as *spacy* gives the same results, except for the "I'm-messing-with-your-mind" pseudo-word, which is broken into parts:

```
['Just', 'in', 'case', 'you', "'re", 'wondering', ',', 'all', 'this',
"I'm", '-',  'messing', '-', 'with', '-', 'your', '-', 'mind', 'chat',
',', 'I', "'ve", 'heard',  'worse', 'on', 'a', 'Friday', 'night', 'in',
'Sheffield', '.']
```

Once again, `you` and `'re`, and `I` and `'ve` are separated contrary to `I'm`, which is considered as a single token.

Chinese is a typical example of language without graphemic word boundaries since sinograms are written without whitespace, except for half-spaces after punctuation marks. Therefore, segmentation of Chinese text is much more difficult than the algorithms used above, where the main issue is whether an apostrophe should be part of a boundary or of a token. Here is an example of a sinogram sequence which can be segmented in two distinct ways:

美国会。

It can be segmented as "**美+国会**"$_{\mathrm{ZH}}$ ⌊America will⌋ or as "**美国+会**"$_{\mathrm{ZH}}$ ⌊US congress⌋ [22, p. 37]. If we attempt to segment it using *spacy*, that is through the following code:

```
import spacy
nlp = spacy.load("zh_core_web_sm")
doc = nlp(""" 美国会.""")
for token in doc:
    print(token.text, token.pos_, token.dep_)
```

then we get the result

```
美国 PROPN nsubj
会 VERB ROOT
. PUNCT punct
```

which means that the segmentation chosen is "US Congress" and not "America will," which is also the automatic translation given by *DeepL* and *Google Translate* and the segmentation given by the Stanford CoreNLP tools. Interestingly, if we consider this sequence of sinograms as being Japanese, then we get the segmentation

```
美 NOUN compound
国会 NOUN ROOT
. PUNCT punct
```

which is a 美+国会. segmentation (the meaning being "beautiful country").

14.2 Zipf's Law

Zipf's law is an empirical statement saying that *in a sufficiently large quantity of text, the frequency of a word is inversely proportional to the word's rank if we order the words by decreasing frequency*.

To verify this law, we took the Whovian corpus (the transcription of 631 episodes[1] containing 258k speech turns, resulting in a total of 2.5M words) and counted words. The results can be seen in Fig. 14.1.

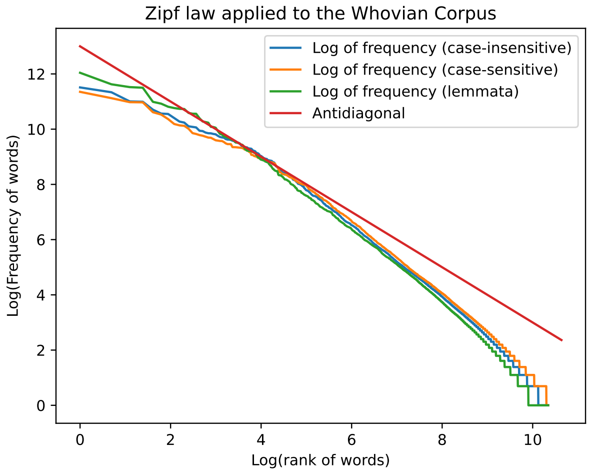


Fig. 14.1 Does Zipf's Law apply to the Whovian universe?

Since the formula is $f \approx \frac{1}{r}$, and we take logarithms in the figure, we would expect an antidiagonal. This is not precisely the case, as the law is rather empirical. The exciting fact of Zipf's law is not that the frequent words are higher ranked, and the rare words (or even the *hapaxes*) have a very low ranking: this is trivial by the very definition of frequency and rank. What is interesting is the fact that this behavior is stable in the sense that the law also applies to all the intermediate cases or moderately frequent and moderately ranked words. Also, we get the same behavior whether we take case-insensitive or case-sensitive word rankings, as well as lemmata (which we obtained after approx. 17 hours of parsing of the corpus by *stanza*).

In the figure, we see that the result is a curve that, in its middle, does have the (affine) antidiagonal as a tangent but is not exactly straight. In other words, frequent words are not as frequent as they could be, and there are not as many rare words as expected. Maybe this is because this corpus is artificial: all dialogs have been written by people who wanted to keep a specific style with a specific vocabulary. The good thing with empirical laws is that when they apply only partially, one can always find explanations to fill the gaps.

[1] From *Chrissie's Transcripts Site* `http://www.chakoteya.net/DoctorWho` (for educational and entertainment purposes only).

Table 14.1 The most frequent "words" of the (case-insensitive) Whovian corpus

R	word	freq.	R	word	freq.	R	word	freq.	R	word	freq.
1	the	98759	16	me	19492	31	oh	13694	46	right	10049
2	you	83841	17	are	18955	32	my	13517	47	get	10009
3	i	60116	18	be	18928	33	don't	13502	48	if	9860
4	to	59199	19	have	18506	34	now	13084	49	so	9303
5	a	44233	20	on	18116	35	here	12386	50	come	9275
6	of	38490	21	do	17304	36	i'm	11855	51	one	9156
7	it	38077	22	it's	16607	37	there	11697	52	out	8858
8	and	32645	23	not	16319	38	know	11525	53	them	8506
9	is	28811	24	all	16163	39	was	11385	54	time	8294
10	that	27933	25	doctor	15208	40	he	11365	55	him	8287
11	what	24413	26	but	15106	41	with	10927	56	as	7940
12	in	23723	27	for	15053	42	they	10907	57	that's	7932
13	we	23437	28	well	14926	43	will	10904	58	us	7660
14	this	20810	29	your	14507	44	can	10830	59	at	7576
15	no	20554	30	yes	14420	45	just	10408	60	think	7538

The most frequent words of the Whovian corpus can be seen in Table 14.1. This table is true, precise, and... boring as hell because all "words" are indeed very common in English. Most of them are grammatical words, and even the ones containing free morphemes seem pretty common as well. The only surprise is the presence of the word "Doctor" at the 25th position: the word "Doctor" is used 15,208 times in the corpus, which is an average of 33.8 times per episode (which is not surprising since it is the main character's name for the last 60 years). The word "think" at the 60th position may also be interesting. If it is more frequent than usual in English, then we can infer that the series involves a lot of thinking. But is this the case? And how can we turn Table 14.1 into something less boring?

To get some *real* information out of the data, we have to compare frequencies and ranks with frequencies and ranks from a more general corpus. Fortunately, there are many of them. The English Wiktionary provides, among other resources, frequency lists of the Project Gutenberg[2] (PG). We compared the rankings and frequencies between the Whovian Corpus and the values in these lists. Frequencies are normalized by dividing by the frequency of the most frequent word (the definite article "the" in both lists).

Table 14.2 is interesting not only because it has colors but because it shows which words in the Whovian corpus behave as in a much larger English corpus (the corpus used to obtain the PG lists contains approx. 1.63G words, that is 652 times more than the Whovian corpus) and which words have special behavior.

The words that have extraordinary behavior are "oh" (PG books do not have as many interjections), "yes" (used mainly in dialogs, and PG books do not have as many dialogs as Dr. Who episodes), and "Doctor." Our hypothesis about the word "think" being more frequent than usual is not valid since other words, such as "get,"

[2] https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/English/Project_Gutenberg

Table 14.2 Comparison of rank and frequencies of the Whovian Corpus with the Project Gutenberg corpus. Positive numbers denote higher values in the Whovian Corpus. Frequencies are relative to the most frequent word of the list (namely "the").

word	Δrank	Δfreq.	word	Δrank	Δfreq.	word	Δrank	Δfreq.	word	Δrank	Δfreq.
the	0	0	me	21	-0.143	oh	2384	-0.138	right	118	-0.092
you	15	-0.741	are	19	-0.134	my	3	-0.078	get	109	-0.091
i	3	-0.399	be	1	-0.091	don't	-33	-0.136	if	4	-0.065
to	0	-0.138	have	6	-0.110	now	41	-0.109	so	-9	-0.044
a	80	-0.427	on	1	-0.092	here	97	-0.112	come	60	-0.078
of	-4	0.213	do	36	-0.145	i'm	-36	-0.120	one	-13	-0.042
it	4	-0.242	it's	-22	-0.168	there	14	-0.083	out	2	-0.056
and	-5	0.201	not	-5	-0.063	know	54	-0.097	them	-10	-0.041
is	4	-0.157	all	7	-0.097	was	-31	0.063	time	15	-0.058
that	-3	-0.086	doctor	991	-0.152	he	-30	0.034	him	-26	-0.013
what	49	-0.218	but	4	-0.083	with	-29	0.026	as	-41	0.044
in	-7	0.069	for	-13	-0.026	they	-9	-0.051	that's	779	-0.078
we	31	-0.193	well	90	-0.136	will	5	-0.069	us	25	-0.056
this	14	-0.139	your	30	-0.117	can	35	-0.088	at	-37	0.013
no	34	-0.168	yes	1127	-0.144	just	104	-0.094	think	73	-0.063

"just," or "here," have even more ranking offsets than "think." Remarkably, words such as "the," "I," "to," "it," "is," "be," "on," "but," "my," "will," "if," and "out" (that's a fifth of the list of words) behave almost precisely in the same way in the two corpora: their rankings differ by less than five units.

As happy as we are to have found such a beautiful free resource as the PG frequency lists, we must still underline the differences between the two corpora: the PG corpus is mainly based on prose and pre-1923 books (for copyright reasons) so that the now extinct word "thy" is ranked no less than 280th! Maybe the first Doctor Who regenerations had an old-fashioned vocabulary,[3]. Still, certainly not the later ones, and pre-1923 vocabulary is used only on specific occasions when time-traveling to previous eras. Overall, having a fifth of the word list being almost perfectly aligned with the PG frequency lists is encouraging.

[3] This is particularly apparent in the 2017 special episode *Twice upon a time* where the 12th Doctor meets the 1st Doctor, who repeatedly utters sexist remarks and fails to acknowledge Bill Potts's homosexuality.

14.3 Stop Words and tfidf

> "To be or not to be" is just a collection of stop words.
>
> John Mueller, *I wouldn't worry about stop words*

Table 14.1 clearly shows that the most frequent words are not the most interesting. By "interesting," we mean words that can provide clues about their context. For example, if we wish to classify a given dialog as belonging to the Whovian corpus or to the *Star Trek* corpus, only the 25th most frequent word, namely "Doctor," actually refers to the former, even though it can very well belong to the latter since there is a *Star Trek* character named "Doctor McCoy." Even worse, if we base our comparison of the two corpora on the most frequent words, then we may get very weak classification performances since they are probably similarly ranked in the two corpora.

Why not remove them? A general trend in NLP is to add very common words to a list of words to be omitted from counting, a *stop list*. Because of this list, they are called *stop words* (in French, they have a much more poetic designation: "mots vides"$_{\text{FR}}$ ⌊empty words⌋). Here is the list of 127 case-insensitive English stop words as defined in the *nltk* package:

i, me, my, myself, we, our, ours, ourselves, you, your, yours, yourself, yourselves, he, him, his, himself, she, her, hers, herself, it, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, should, now.

Shall these be systematically removed? Well, ostracism is always a delicate and risky operation: some stop words may be necessary for a given task, or, even worse, some specific instances of stop words may be important while others may not. As the epigraph to this section says, "To be or not to be" is a collection of words commonly considered as stop words, and removing them from the Shakespearian corpus would leave Hamlet speechless before his skull.

A technique that can shed more light on the "stop-wordness" of specific word instances in specific documents of a corpus is the *tfidf* weight. The initialism "tfidf" expands to "text frequency times inverted document frequency." In its simplest form, tfidf is defined as follows:

$$\mathsf{tfidf}_C(w_0, d_0) := \frac{\#\{w_0 \in d_0\}}{\#\{d \in C \mid w_0 \in d\}},$$

where w_0 is a given word, d_0 a given document and C a corpus. The numerator (the "tf" part) is the number of occurrences of w_0 in document d_0. The denominator (the "idf" part) is the number of documents d in the corpus that contain w_0. The idea is that "the more frequent w_0 is in the document d_0, the higher its tfidf" and "the more documents contain w_0, the lower its tfidf". Stop words are exactly in the latter case: chances are *every* document in the corpus contains "the," therefore its tfidf will be very low.

Let us take an example. We will calculate the tfidf of words of the Dr. Who *Heaven Sent* S35E11, with respect to the complete Whovian corpus. We will use the tfidf implementation of the *sklearn* package. Here is the code:

```
1  import xml.sax
2  import sys,re
3  from sklearn.feature_extraction.text import TfidfVectorizer
4
5  EPISODES=[]
6  EPISODE_NAMES=[]
7
8  class CustomContentHandler(xml.sax.ContentHandler):
9      def __init__(self):
10         self.contents = ''
11     def startElement(self, tagName, attrs):
12         global EPISODES_NAMES
13         if tagName == 'episode':
14             EPISODE_NAMES.append(attrs['id'])
15             self.contents=""
16     def endElement(self, tagName):
17         global EPISODES
18         if tagName == 'episode':
19             EPISODES.append(self.contents)
20         elif tagName == 'p':
21             self.contents += '\n'
22     def characters(self, chars):
23         self.contents += re.sub(r'[.,?!]',r'',chars.lower())
24
25 handler = CustomContentHandler()
26
27 xml.sax.parse('dr-who.xml', handler)
28
29 vectorizer = TfidfVectorizer(stop_words='english')
30 X = vectorizer.fit_transform(EPISODES)
31 words=vectorizer.get_feature_names_out()
32
33 episode_index=EPISODE_NAMES.index("Heaven Sent")
34 tfidfs=[(words[i],X[(episode_index,i)]) for i in range(len(words)) \
35     if X[(episode_index,i)] > 0]
36 tfidfs=sorted(tfidfs,key=lambda x: x[1],reverse=True)
37 print("Highest tfidfs:",tfidfs[:10],"Lowest tfidfs:",tfidfs[-10:])
```

☞ 367 The corpus is contained in an XML file. We use the SAX package to parse it: episodes are contained in `episode` elements, the name of which is contained in an `id` attribute. On line 23, we convert the textual content we read into lowercase and strip punctuation. Then, on line 19, we store the accumulated text in the global `EPISODES` list and similarly store the episode's name into the `EPISODES_NAMES` list, on line 14. We parse the file on line 27, instantiate a tfidf "vectorizer" online 29, and vectorize the `ELEMENTS` list on line 30: every entry of the list is considered as a document. We store words into `words` on line 31. We are interested in a specific episode, so on line 33, we extract the index of that episode. Using that index, we create `tfidfs`, a list of tuples, each tuple containing a word (with nonzero tfidf value) and its tfidf value. Finally, we sort this list in decreasing order of the tfidf value and print the ten highest and ten lowest words.

Here are the results for the *Heaven Sent* episode:

Highest tfidf values				Lowest tfidf values			
clara	0.392	scared	0.139	er	0.007	sure	0.007
hybrid	0.220	bird	0.121	haven	0.007	wait	0.007
shepherd	0.205	years	0.116	mind	0.007	thought	0.007
argh	0.161	punch	0.111	stay	0.007	mean	0.007
know	0.142	time	0.109	listen	0.007	good	0.006

Incredible as it may seem, "Clara" is the episode's highest-ranked word. Indeed, "Clara" appears 18 times in it, but one could argue that "Clara" also appears in other episodes, and therefore its tfidf should not be that high. We investigated: "Clara" appears 769 times in the corpus, and its use is concentrated in 40 episodes. What counts for the tfidf calculation is the number of episodes in which the name appears, namely 40, which is only 9% of the corpus. The combination of relatively high frequency in this episode (tf) and a relatively small number of episodes where the name appears (idf) has elevated this name to the top.

The situation is more straightforward for "hybrid" and "shepherd": the former appears seven times in this episode and only 12 other episodes, and the latter appears six times in this and in eight other episodes. On the other extremity of the range of tfidf values, we have the words "er," "haven," "mind," "stay," "sure," "wait," "thought," "mean," and "good." To give only an example, the word "good" is a hapax in this episode and appears in 447 other episodes. In virtually all episodes, these are the extreme conditions of low tf and high idf. Words with low tfidf values are potential stop words for the specific document.

Tfidf is far from perfect. A significant drawback, as we saw, is the fact that the idf part does not take global frequency into account but only the number of documents in which the word appears: whether it appears 1,000 times or a single time in each of the documents (other than the current one), the value is the same. Another drawback is that tf is not normalized by document size: larger documents tend to have higher tf values even when the density of the word in the text is the same or even lower.

14.4 Collocations

> Tradition says that one linguist, who had heard enough of frequency counts, predicted ironically that when the counters had grown tired of tabulating single items, they would begin to count words by pairs.
>
> David G. Hays, *Introduction to Computational Linguistics*

A *collocation* is a sequence of words frequently occurring together. We have discussed them in §4.9. This section will show how to calculate them in a corpus.

The blatant first attempt is to count the frequency of word pairs. The corpus must be prepared so punctuation is not part of the word tokens. An easy solution is to insert whitespace before punctuation marks so that they become individual tokens. Punctuation must not be removed. Otherwise, we will get parasitic pseudo-collocations that do not exist in real life (collocations are not supposed to contain punctuation). The following code from the *nltk* package allows filtering so that pairs with punctuation tokens are not taken into account:

```
1  import io
2  from nltk.collocations import BigramCollocationFinder
3  from nltk.metrics import BigramAssocMeasures
4
5  punct=set([".",",","?",";",":","!"])
6
7  f=io.open("dr-who.txt",mode="r",encoding="utf-8")
8  words=[]
9  for l in f:
10     words.extend(l.lower().split())
11 f.close()
12
13 bigram_collocation = BigramCollocationFinder.from_words(words)
14 bigram_collocation.apply_word_filter(lambda w: w in punct)
15
16 print(bigram_collocation.nbest(BigramAssocMeasures.raw_freq, 50))
```

On line 5, we define the set of punctuation marks we want to ignore, line 7 we read the corpus (it is a version of the Whovian corpus where we have stripped all markup). On line 8 we store all word tokens into a list `words`; on line 13 we create a collocation instance out of this corpus; on line 14, we apply the filter (pairs with at least one token belonging to `punct` are omitted), and on line 15 we calculate the fifty most frequent collocations using raw frequency as a measure.

Here are the results:

> "of the," "in the," "are you," "do you," "to the," "going to," "i know," "come on," "the doctor," "all right," "you know," "i think," "to be," "this is," "on the," "you are," "i have," "out of," "of course," "it is," "is it," "have to," "i am," "to do," "if you," "i can," "it was," "you have," "i was," "thank you," "what are," "what is," "want to," "and the," "what do," "did you," "you can," "must be," "will be," "is the," "for the," "to get," "you will," "you think," "got to," "the tardis," "and i," "have you," "we have," "from the"

Needless to say these collocations are disappointing: most words are grammatical, so these collocations do not contribute to an understanding of the corpus. Even when one of the two words is significant, the other is just a grammatical word: this happens in “the Doctor” at the 9th position and “the Tardis” at the 46th position. We must remove stopwords by adding them to the filter, which now becomes:

```
from nltk.corpus import stopwords
stopset = set(stopwords.words('english'))
bigram_collocation.apply_word_filter(lambda w: len(w) < 3 \
    or w.lower() in stopset or w in punct)
```

The results have changed dramatically:

“get back,” “come along,” “come back,” “time lord,” “time lords,” “never mind,” “one thing,” “get away,” “looks like,” “look like,” “years ago,” “long time,” “well done,” “good luck,” “old man,” “must get,” “one day,” “young man,” “last time,” “must find,” “make sure,” “hundred years,” “thousand years,” “miss grant,” “better get,” “dear boy,” “never seen,” “even know,” “something else,” “think going,” “little bit,” “take care,” “get rid,” “human race,” “still alive,” “much longer,” “good idea,” “last night,” “coming back,” “police box,” “human beings,” “time machine,” “five minutes,” “think better,” “could get,” “two minutes,” “really think,” “never get,” “would like,” “young lady”

Some collocations in this list are very interesting, such as “time lord(s)” at the 4th and 5th position, “hundred years” and “thousand years” at the 23rd and 24th position, “miss Grant” at the 25th position[4], “human race,” and “time machine.” One may wonder why “Miss Grant” is the only named entity that emerges from this search. The collocation “Miss Grant” occurs 183 times in the corpus and is the most frequent pair starting with “Miss,” here are the other ones:

Miss Smith (106 times)
Miss Shaw (71 times)
Miss Foster (37 times)
Miss Wright (35 times)
Miss Kelly (33 times)
Miss Hawthorne (31 times)
Miss Jones (29 times)
Miss Kizlet (25 times)
Miss Oswald (16 times)

On the other hand, “Grant” is preceded 30 times by “Jo” and otherwise only by stop words or unfrequent words.

Using raw frequency as a measure gives us the most frequent collocations, but not necessarily the *strongest* ones. We can define AB as a *strong collocation* if AB is frequent and AB' (and $A'B$) is unfrequent for any A' (or B'). This means that we are interested not so much in the frequency of the collocation but in its solidity. A typical example of a strong collocation is the name “Lethbridge Stewart”: this pair appears 79 times in the corpus, and “Lethbridge” is *always* followed by “Stewart.” (On the other hand, “Stewart” is preceded 16 times by “Kate” and a few times by “Brigadier,” “Commander,” etc.)

---

[4] Jo Grant was a companion of the 3rd Doctor, she appeared in 77 episodes, starting with the *Terror of the Autons* S8E1. She posed naked with a Dalek in *Girl Illustrated*, in 1978.

To obtain a measure of collocation strength, we will use conditional probabilities. Using an approach from Manning & Schütze [17, pp. 172–175], let w_1 et w_2 be two given words, let p_1 be the probability $P(w_2 \mid w_1)$ of having the pair $w_1 w_2$ when we know that the first word is w_1, and p_2 the probability $P(w_2 \mid \neg w_1)$ of having a pair Xw_2 when we know that the first word X is not w_1.

Suppose we now count the frequencies of w_1, w_2, and $w_1 w_2$, in a corpus of N words. Let us call these numbers c_1, c_2 and c_{12}. Let us take all the pairs with w_1 as the first element. There are c_1 of them (we consider that w_1 is never the last word of a sentence). Let us take the two following events:

1. c_{12} among the c_1 pairs starting with w_1 are $w_1 w_2$;
2. $c_2 - c_{12}$ among the $N - c_1$ pairs not starting with w_1 have w_2 as second element.

These two events are independent since they deal with different elements (pairs starting with w_1 for the first event, pairs not starting with w_1 for the second event). Therefore, the probability that both occur is equal to the product of the probability that each one occurs. To calculate the probability of (1), we have to find the probability of a given sequence of pairs of length c_1 containing c_{12} pairs $w_1 w_2$. This probability is

$$p_1^{c_{12}}(1-p_1)^{c_1-c_{12}},$$

where we have defined p_1 as being the probability of $w_1 w_2$. This is the probability of a specific arrangement. To get the probability of event 1 for all arrangements, we multiply by the number of possible arrangements:

$$\binom{c_1}{c_{12}} p_1^{c_{12}}(1-p_1)^{c_1-c_{12}}.$$

Similarly, the probability of event (2) is

$$\binom{N-c_1}{c_2-c_{12}} p_2^{c_2-c_{12}}(1-p_2)^{N-c_1-(c_2-c_{12})},$$

where p_2 is the probability of having pairs ending with w_2, but not starting with w_1. By multiplying the probabilities of the two events we get the probability of having c_1 pairs starting with w_1, c_2 pairs ending with w_2 and c_{12} pairs $w_1 w_2$:

$$\binom{c_1}{c_{12}} p_1^{c_{12}}(1-p_1)^{c_1-c_{12}} \binom{N-c_1}{c_2-c_{12}} p_2^{c_2-c_{12}}(1-p_2)^{N-c_1-(c_2-c_{12})}. \tag{14.1}$$

We don't know the probabilities p_1 and p_2, but we can approximate them by our measurements:

$$p_1 = P(w_2 \mid w_1) = \frac{P(w_1 w_2)}{P(w_1)} \approx \frac{c_{12}/N}{c_1/N} = \frac{c_{12}}{c_1}, \tag{14.2}$$

$$p_2 = P(w_2 \mid \neg w_1) = \frac{P(\neg w_1 w_2)}{P(\neg w_1)} \approx \frac{(c_2 - c_{12})/N}{1 - c_1/N} = \frac{c_2 - c_{12}}{N - c_1}. \tag{14.3}$$

Inserting this into (14.1), we get a long and complex formula. At this point, the reader may wonder how we will compare the collocation strength of various pairs.

Formula (14.1) is the general case of a pair that can be more or less collocated. The trick is to compare this value with the one we would get if w_1 and w_2 were not collocated at all, in the sense that the probability of having w_2 knowing that the previous word is w_1 is the same as having w_2 in general, i.e., the choice of w_2 is not influenced by the presence of w_1 before it. In that case, we have

$$P(w_2 \mid w_1) = P(w_2 \mid \neg w_1) = P(w_2),$$

i.e., $p_1 = p_2 = p$, where p is the probability of having the word w_2, which we can approximate by $\frac{c_2}{N}$.

Let us call $L(H_2)$ the formula (14.1) where we have approximated p_1 and p_2 as in (14.2) and (14.3). L stands for the likelihood of our counts and H_2 for the hypothesis that w_1 and w_2 are dependent. And let us call $L(H_1)$ the formula (14.1) where we have replaced p_1 and p_2 by c_2/N. Here H_1 stands for the hypothesis of independence of w_1 and w_2.

Our goal is to quantify the dependence of w_1 and w_2. We seek a measure that is 0 when w_1 and w_2 are independent and positive otherwise. Let us take

$$\log \frac{L(H_2)}{L(H_1)}$$

and call it the *log of likelihood ratio*. This measure is strictly positive when the choice of w_2 is influenced by the presence of w_1 and zero otherwise.

The code is similar to the one using raw frequencies. We only change the name of the measure:

```
print(bigram_collocation.nbest(BigramAssocMeasures.likelihood_ratio, 50))
```

Here are the results:

> "come along," "time lords," "get back," "time lord," "years ago," "looks like," "never mind," "miss grant," "good luck," "come back," "thousand years," "young man," "hundred years," "one thing," "get away," "human beings," "old man," "police box," "dear boy," "human race," "long time," "get rid," "lethbridge stewart," "new york," "one day," "much longer," "make sure," "young lady," "solar system," "five minutes," "take care," "last night," "well done," "miss smith," "old chap," "little bit," "captain yates," "something else," "sarah jane," "ten minutes," "never seen," "prime minister," "still alive," "last time," "great deal," "five hundred," "black hole," "look like," "ever since," "sonic screwdriver"

They are very satisfying: not only have we kept "time lord(s)," "human race," and "miss Grant," but now we have also "police box" (18th position), "Lethbridge Stewart," "solar system," "Miss Smith," "Captain Yates," "Sarah Jane," "prime minister," "black hole," and even, at the 50th position, the "sonic screwdriver."

These results can be improved even further if we impose restrictions on the POS tags of words, for example, by avoiding adverbs or numerals.

14.5 N-grams

In the previous section, we considered pairs of words and investigated whether a pair's second word depends on the pair's first word. We can generalize this by looking at sequences of n words and calculating the probability that a specific $n + 1$-st word follows them. And once we have calculated the probabilities, we can guess the $n + 1$-st word by taking the candidate with the highest probability. This is what mobile devices do when they autocomplete user input. It is funny (and sometimes worrisome) to look at Google autocompletion, such as

why does my da → why does my dad hate me

can a human → can a human survive being skinned alive

did the Doctor → did the Doctor love Amy/Rose/Clara

you smell like → you smell like beef and cheese

authors are → authors are highly esteemed brainly

Of course, these sentences do not reflect language usage in general. They are just the most probable sentences in the specific corpus of Google queries.

We will call an assignment of probabilities to sentences a *language model*. The idea is that even though we can utter *any* sentence, including those built by *exquisite corpses*[5], we tend to utter sentences that fulfill our communication goals, making these sentences more probable than others. The language model of a language allows the calculation of the probability of any sentence so that the most probable next word can be inferred—at least in theory.

To calculate a language model, the approach that comes to mind first is to count all n-grams (for $n \geq 1$) so that the more frequent a given sequence of words $w_1, \ldots, w_n$ is, the higher the probability we assign to it in the language model.

There are two problems, though: (1) how many times will a given n-gram appear so that we can count occurrences and assign a probability to it? (2) a text of N words contains $(N - n)$ n-grams, to what extent do they cover all possible (grammatical) combinations of the N words?

Here is a test. We stripped all markup from the Whovian corpus document, extracted all 5-grams, and removed those containing punctuation. 1,011,655 5-grams remained. Then, we searched 5-grams that occurred more than once. Here are the fifteen most frequent 5-grams:

what are you going to (208 times)
are you going to do (203 times)
what are you doing here (173 times)
no no no no no (170 times)
what are you talking about (158 times)
are we going to do (148 times)

---

[5] An exquisite corpse is the result of a game, a text built collectively by following rules that introduce some randomness, for example, the fact that each player sees only the last word written by a previous player. The technique was introduced by the surrealists in the 1920s. The technique's name originates from a sentence obtained through it: "le cadavre exquis boira le vin nouveau"$_{\text{FR}}$ ⌊the exquisite corpse will drink the new wine⌋.

what are we going to (142 times)
to get out of here (134 times)
let's get out of here (101 times)
got to get out of (69 times)
do you want me to (62 times)
we've got to get out (62 times)
as a matter of fact (61 times)
there's nothing we can do (59 times)
what do you make of (57 times)

There are only 474 5-grams that occur more than ten times, so their frequency is statistically significant, among more than a million grammatical 5-grams and 4.44×10^{22} potentially nonsensical 5-grams. In other words, the longer n-grams we take, the more sparse the data will be, making the probability calculation inefficient.

This is why most of the work done with n-grams is restricted to $n = 1, 2, 3$ and eventually 4. To illustrate the notion of language model, let us attempt an autocomplete on a specific corpus. In the XML Whovian corpus, we have the information on the speaker of each speech turn. We will keep only utterances of the Doctor. Based on this corpus, we will build a language model of unigrams, bigrams, trigrams, and tetragrams and try some autocompletes.

Here is the code for $n = 1, 2, 3, 4$:

```
import io,nltk,random

with io.open('doctor-doctor.txt',mode="r",encoding="utf-8") as file:
    text = file.read().lower().rstrip().split()

bigrams=nltk.ngrams(text,2)
cfd2=nltk.ConditionalFreqDist(bigrams)

trigrams=nltk.ngrams(text,3)
condition_pairs3 = (((w0, w1), w2) for w0, w1, w2 in trigrams)
cfd3=nltk.ConditionalFreqDist(condition_pairs3)

tetragrams=nltk.ngrams(text,4)
condition_pairs4 = (((w0, w1, w2), w3) for w0, w1, w2, w3 in tetragrams)
cfd4=nltk.ConditionalFreqDist(condition_pairs4)

def autocomplete1(num=15):
    print("n=1: ")
    for i in range(num):
        j=random.randrange(0,len(text))
        print(text[j]," ",end="")
    print("")

def autocomplete2(cfdist, word_1, num=15):
    print("n=2: ")
    for i in range(num):
        print(word_1," ",end="")
        word_1 = cfdist[word_1].max()
    print("")

def autocomplete3(cfdist, word_1, word_2, num=15):
```

```
32      print("n=3: ",word_1," ",end="")
33      for i in range(num):
34          print(word_2," ",end="")
35          word_3 = cfdist[(word_1,word_2)].max()
36          (word_1,word_2)=(word_2,word_3)
37      print("")
38
39  def autocomplete4(cfdist, word_1, word_2, word_3, num=15):
40      print("n=4: ",word_1," ",word_2," ",end="")
41      for i in range(num):
42          print(word_3," ",end="")
43          word_4 = cfdist[(word_1,word_2,word_3)].max()
44          (word_1,word_2,word_3)=(word_2,word_3,word_4)
45      print("")
46
47  autocomplete1()
48  autocomplete2(cfd2,"i'm",)
49  autocomplete3(cfd3,"i'm","a")
50  autocomplete4(cfd4,"i'm","a","time")
```

Some explanations: the file `doctor-doctor.txt` contains only the speech turns of the Doctor, a total of 65,795 speech turns (out of 258,241 in total) and 728,642 words (which is a bit less than the King James Bible, which contains 783,137 words, but still a significant amount of words). On line 4, we read the whole file, convert it to lowercase, remove end-of-lines, and split at whitespace so that `text` is a list containing all word tokens uttered by the Doctor.

The *nltk* method `ngrams(text, n)` returns a list of tuples containing all n-grams. It works like a sliding window for length n. The `ConditionalFreqDist` method builds a dictionary with $(n-1)$-grams as keys. For each key, the value is again a dictionary containing all attested n-th elements of n-grams as keys and their frequencies as values. For example, considering there is a tetragram "Run like the wind," then in `cfd4` there will be a key `("Run", "like", "the")` with the value `("wind": 1)` (in fact it will even have the frequency value 2, since "Run like the wind" has been uttered twice: in October 1967 (S5E2) and in September 2012 (S33E2)).

Lines 10 and 14 may seem intriguing: in fact, `ConditionalFreqDist` expect 2-tuples, the first element of which is the key and the second the value. On line 7, `bigrams` already has the structure of 2-tuples that the method expects, but on line 9, `trigrams` is a list of 3-tuples, on which `ConditionalFreqDist` chokes. Line 10 converts `trigrams` into the right format, and so does line 24 for `tetragrams`.

The `autocomplete1` function chooses random words. For the other autocompletes, as seen on lines 28, 35 and 43, we always take the most frequent continuation so that the result is deterministic. Here are the results for "I'm," "I'm a," and "I'm a time":

$n = 1$: the no like cover we . the yes now and a be got around when

$n = 2$: i'm not a bit of the tardis . i don't know . i don't know

$n = 3$: i'm a time lord . i don't know . i don't know . i don't know

$n = 4$: i'm a time lord . i beg your pardon . i thought you were going to say

The deterministic nature of the code causes the endless loops of "I don't know" when $n = 2$ or $n = 3$. By applying the method `.most_common(10)` instead of `.max()`, we get the (at most) 10 possible continuations, together with their frequencies:

i'm

a

time → ('lord', 23), ('traveller', 6)

lord → ('.', 42), ('from', 3), ('could', 2), ('has', 2), ('i', 2), ('not', 2), ('and', 2), ('in', 2), ('can', 2), ('jo', 1) . ('i', 5), ('i'm', 4), ('oh', 2), ('yes', 2), ('you', 2), ('that's', 2), ('come', 2), ('well', 2), ('we', 2), ('you're', 2)

i → ('beg', 1), ('told', 1), ('suspect', 1), ('am', 1), ('can't', 1), ('just', 1), ('can', 1)

beg → ('your', 19), ('you', 2), ('the', 1), ('to', 1)

your → ('pardon', 24)

pardon → ('.', 10), ('oh', 2), ('sir', 2), ('ah', 2), ('extraordinary', 1), ('awful', 1), ('young', 1), ('but', 1), ('look', 1), ('i'm', 1)

. → ('i', 3), ('that's', 2), ('part', 2), ('everything', 1), ('the', 1), ('what's', 1)

i → ('thought', 1), ('do', 1), ('beg', 1)

thought → ('you', 30), ('i', 18), ('the', 9), ('i'd', 8), ('so', 7), ('it', 7), ('you'd', 6), ('we'd', 5), ('they', 4), ('if', 3)

you → ('were', 27), ('said', 17), ('might', 9), ('meant', 2), ('didn't', 2), ('couldn't', 1), ('did', 1), ('ought', 1), ('took', 1), ('would', 1)

were → ('going', 5), ('dead', 4), ('just', 3), ('a', 2), ('in', 2), ('rejecting', 2), ('drowned', 1), ('doing', 1), ('never', 1), ('the', 1)

going → ('to', 14), ('straight', 2)

to → ('say', 3), ('attack', 2), ('show', 2), ('make', 2), ('see', 2), ('drop', 2), ('be', 1), ('think', 1), ('tell', 1), ('help', 1)

say → ('that', 4), ('it', 2), ('.', 2), ('any', 1), ('how', 1), ('am', 1), ('retreat', 1)

Using this method and choosing elements randomly, we avoid endless loops and get more natural results. Here is the code for the random choice:

```
word_4 = random.choice([x for (x,y) in \
   cfdist[(word_1,word_2,word_3)].most_common()])
```

And here is the first result we obtained:

i'm a time traveller or at least be interested in . i met you somewhere before oh

which is indeed remarkable.

How can we evaluate the results of our autocomplete task? If the goal is to obtain clever, imaginative, or poetic autocompletion, then evaluation should better be manual. But if we aim to obtain autocompletion that conforms with the Whovian corpus, i.e., if we want to automatically obtain what the Doctor would say in a given situation, we can perform an objective evaluation. Here is how: We will shuffle the corpus with respect to speech turns (so that speech turns from different eras get all mixed), and then we will perform a 10-fold split over the corpus. This will give us ten pairs of training and test corpora, so the merge of the 10 test corpora is the original corpus.

Then, we will calculate conditional frequency distributions of bigrams, trigrams, and tetragrams for each training set. For each $(n-1)$-gram of them, we will keep the complete distribution, that is, all potential completions. Then, we will calculate the tetragram conditional frequency distribution for the test set corresponding to a given training set. For each triple (w_1, w_2, w_3), which is a key in this distribution, we will check whether it exists in the training set. If yes, we will compare the distribution of (w_1, w_2, w_3) in the test set with the one in the training set by calculating the ratio of the size of their intersection with the size of the distribution in the test set. For example, if there are n quadruplets "i'm a time x" (for any x) in the test set, out of which there are m quadruplets "i'm a time y" in the train set, then we will keep the number m/n.

If the (w_1, w_2, w_3) from the test set does not exist in the train set, we will look at (w_2, w_3) among the $(n-1)$-sequences of the trigrams of the train set, and if there is no $(w_2, w_3, *)$ trigram, we will check whether there is a $(w_3, *)$ bigram.

Once we have done these calculations for every pair of train and test sets, we take the average of the results. This is called 10-fold cross-validation and has the advantage that every element of the corpus appears precisely once in a test set, so the bias due to the choice of the test set is very small.

Here are the results we obtained:

	$(n-1)$-grams attained	ratio of completions found
tetragrams	36.20%	33.67%
tetragrams+trigrams	77.55%	40.54%
tetragrams+trigrams+bigrams	98.40%	45.90%

which means that if we train our algorithm on the tetragrams of 90% of the corpus, we will have an autocompletion for 98.4% of the trigrams of the remaining 10%. In 36.2% of the cases, this autocompletion will be based on tetragrams from the training set and, therefore, will be of good quality. In 77.55% of cases, it will be based on tetragrams and trigrams, and otherwise, it will be based on n-grams for $n = 2, 3, 4$. The autocompletion we obtain from the training corpus cover between 33.67% and 45.90% of the completions occurring in the test set. This means that between a third and almost a half of the set of fourth words of tetragram starting with (w_1, w_2, w_3) in the test set, already occur as fourth words of tetragrams starting with (w_1, w_2, w_3) in the training set.

14.6 Vector Semantics

> Now the linguist's "meaning" of a morpheme (or of a combination thereof, of any complexity up to a complete utterance, which might even be a whole book or poem) is by definition the set of conditional probabilities of its occurrence in context with all other morphemes
>
> Martin Joos, *Description of Language Design*

In the previous section, we observed that n-grams, for n not too big, tend to re-appear so that guessing the n-th word from the $(n-1)$ words preceding it is not entirely hopeless. Being able to guess with some success a word by those preceding it also means that the presence of a word in a sentence depends on its surrounding words.

Let us take this phenomenon the other way around: suppose we have a "context," a group of words obtained from a sentence from which we removed a word. If we take

I eat _ for breakfast

the words that can reasonably fill the gap are "eggs," "cornflakes," "bacon," etc. In the Whovian corpus, we do find the gap filled with the word "danger" (*Arachnids in the UK* S37E4), but this is an "outlier" since, in most cases, we expect the missing word to refer to some food.

Similarly to the epigraph of this section, the *distributional hypothesis* [12] asserts that *words that occur in similar contexts have similar meanings*. The problem with this hypothesis is that the adjective "similar" can be very precise when we talk about contexts (for example, we can define a context as a sequence of n words surrounding _) but is very vague regarding meanings. What are "similar meanings"? Synonyms, provided they exist, can claim to have a similar meaning to each other, but in the example above, "eggs" is certainly not a synonym for "cornflakes." So in what sense are "eggs" and "cornflakes" (or even "danger") *similar*, as says the distributional hypothesis?

Let us amend the distribution hypothesis to avoid this problem: we claim that *words that occur in similar contexts have related meanings*. "Eggs" is related to "cornflakes" and vice-versa since both are ingredients of what is called breakfast in some places, and "eggs" is (remotely) related to "danger" because of the metaphor used by the Doctor. Indeed, the former relatedness (between "eggs" and "cornflakes") is much stronger than the latter (between "eggs" and "danger"). This difference in strength is mirrored by the statistics of fillers of the "I eat _ for breakfast" sentence: the word "eggs" occurs much more frequently (at least in texts referring to the US or the UK) than "danger."

Semantic relatedness is the property of words (or nominal phrases) occurring in similar contexts, while *semantic similarity* is the property of having close meanings. An example of similar words is "car" and "automobile." They have the same mean-

ing but are not entirely interchangeable[6] A typical example of a pair of strongly related words is "car" and "gasoline." Not only are they not synonyms, antonyms,
☞ 143 meronyms, or any other kind of -nyms, but if you look into WordNet, they belong to entirely different branches: the first is a metallic artifact, and the second is a liquid. But they are strongly related since when we talk about cars, we often talk about gasoline.

Relatedness is an essential property because it allows us to compare words and units that contain them: sentences, paragraphs, tweets, emails, documents, etc.

Given two words, how can we calculate semantic relatedness? Furthermore, how can we do it quickly and with minimal CPU use since comparing two documents may require a lot of word-pair relatedness calculations? It didn't take long for computer scientists to realize that checking whether two words have the same neighbors can be represented by a scalar product: if we have five words $\{a, b, c, d, e\}$ in my vocabulary and, in some context, a has b and d as neighbors, we can write this as $(0, 1, 0, 1, 0)$; if e has a and d as neighbors, we get the vector $(1, 0, 0, 1, 0)$. If they have one word as a common neighbor, we can declare that their relatedness is 1 (or $1/2$ if we want to normalize by the average number of neighbors of the two words). An algebraic means to state that they have "one common neighbor" is to take the scalar product of the vectors, and scalar products are very fast operations for the computer.

This finding led computer scientists to the question: "How can we store information about words or nominal phrases in vectors so that semantic relatedness can be obtained simply by a scalar product?" This approach is called *vector semantics*.

14.6.1 LSA and ESA

One of the first attempts to calculate a vector representation of words was *Latent Semantic Analysis* (LSA) [15]. Let us take a corpus of documents d_* containing words w_*. By observing the presence of words in the documents, one obtains a matrix C called "term-document matrix," the rows of which represent words and the columns, documents. Whenever we observe that a word w_i appears in document d_j, the cell $c_{i,j}$ of C at row i and column j is set to 1.

The matrix C can be quite large; for example, for the Whovian corpus, if we consider episodes as "documents," we will need 631 columns (for the 631 episodes) and 39,220 rows (for the 39,220 tokens obtained when we convert to lower case, remove all punctuation except periods, and split by whitespace). To reduce the dimensions of this matrix, LSA uses the technique of *Singular Value Decomposition* (SVD), which consists in decomposing any $m \times n$ matrix C into a product

$$C = UDV,$$

[6] See https://perma.cc/RA5N-FV4C for a lovely discussion on the difference between these words.

where U is $m \times m$, D is $m \times n$ with values only on the diagonal, and V is $n \times n$. Furthermore, the values of D are positive, and U and V can be chosen so that the values of D are in decreasing order. The trick now is to say that if we consider C as a transformation of m-dimensional vectors into n-dimensional vectors, then the "essential part" of the transformation is concentrated in the first values of the diagonal of D. Therefore we replace D by a square $r \times r$ diagonal matrix D', where $r \ll \mathtt{min}(m, n)$. We call U' the first r columns of U and V' the first r rows of V. If we then write

$$C' = U'D'V',$$

we get a matrix C' of low dimensionality, which is quite close to C when considered a transformation. The matrix U' assigns to each word a vector (row of U'), which synthesizes the presence of the word in a small number of "virtual documents." These "virtual documents" are called *topics*, and their detection is a task called *topic modeling*. Similarly, V' assigns topics to documents. U' is therefore a term-topic matrix and V' is a topic-document matrix.

By taking the scalar products of vectors in U' we can compare words. By taking spatial cosine distances of vectors in V' we can compare documents.

Let us give an example of the use of LSA. We will take the Whovian corpus with episodes as "documents," and we will search for the relatedness of the names of the seven companions of the Doctor. As LSA considers the co-presence of these tokens in the same episodes, we should normally find a small spatial distance for Amy and Rory and a higher distance between other companions. Here is the code:

```
import io,re,numpy
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
from scipy import spatial

WORDS={}
max_word=0
EPISODES=[]

f=io.open("dr-who-for-lsa.xml",mode="r",encoding="utf-8")
for l in f:
    if (re.match(r'<episode',l)):
        EPISODES.append([])
    else:
        ltab = l.rstrip().split()
        for w in ltab:
            if w not in WORDS.keys():
                WORDS[w]=max_word
                max_word += 1
            EPISODES[-1].append(WORDS[w])
f.close()

Xdense=numpy.zeros((len(WORDS.keys()),len(EPISODES)))

counter=-1
for E in EPISODES:
    counter += 1
```

```
28         for windex in E:
29             Xdense[windex,counter]=1
30
31     X=csr_matrix(Xdense)
32     svd=TruncatedSVD(n_components=100, n_iter=7)
33     Xnew=svd.fit_transform(X)
34
35     test=["clara","amy","rory","yaz","rose","donna","martha"]
36
37     for A in test:
38         Avec = Xnew[WORDS[A],:]
39         print(A," ",end="")
40         for B in test:
41             Bvec = Xnew[WORDS[B],:]
42             res=spatial.distance.cosine(Avec,Bvec)
43             print(f'{res:.3f} ',end="")
44         print("")
```

On lines 10–21, we build a dictionary `WORDS` that connects words to their indices, and a list `EPISODES` which contains a list for each episode, containing the indices of the words in the episode. On lines 23–29, we set the cell (i, j) of a dense matrix `Xdense` equal to 1 whenever a token i appears in episode j. On line 31, we convert this matrix into a sparse one. On lines 32–33, we apply SVD to this matrix, lowering the dimensions to 100. Finally, on lines 35–44, we take the spatial distance cosine of vectors corresponding to the companions' names (line 35) and print the result into a table. Here is the result:

	Clara	Amy	Rory	Yaz	Rose	Donna	Martha
Clara	0.000						
Amy	0.515	0.000					
Rory	0.573	0.026	0.000				
Yaz	0.945	0.989	0.976	0.000			
Rose	0.861	0.838	0.847	0.913	0.000		
Donna	0.971	0.887	0.896	0.992	0.461	0.000	
Martha	0.910	0.870	0.874	0.961	0.436	0.278	0.000

As expected, Amy and Rory are very close (0.026). We also observe that Martha and Donna are pretty close (they meet in episode *The Sontaran Stratagem* S30E4), Rose is (moderately) close to Martha and Donna (their names appear together in a few episodes), Clara is close to Amy and Rory (the Doctor mentions Amy to Clara in *Deep Breath* S34E1). And, of course, every word is at zero distance from itself.

The problem with LSA is its lack of explicability. As Gabrilovich & Markovitch put it,

> LSA is essentially a dimensionality reduction technique that identifies some most prominent dimensions in the data, which are assumed to correspond to "latent concepts." Meanings of words and documents are then compared in the space defined by these concepts. Latent semantic models are notoriously difficult to interpret, since the computed concepts cannot be readily mapped into natural concepts manipulated by humans. [9, p. 1610]

Therefore Gabrilovich & Markovitch developed a technique called *Explicit Semantic Analysis* (LSA) as an explicable version of LSA [9, 11]. The main differences are:

1. the "term-document" matrix contains not units but tfidf values of the terms in the documents;
2. the corpus is Wikipedia.

By using Wikipedia, Gabrilovich & Markovitch were confident that co-presence in the same documents was equivalent to relatedness. By using tfidf, they attained better precision since the importance, and not only the presence, of a given term for a given document is considered.

Explicability is guaranteed since there is no matrix reduction, and every word corresponds to a vector in the space of Wikipedia pages. Every coordinate of this vector can be explained by the importance of the term (represented by its tfidf) for the specific Web page. Two years after the original paper by Gabrilovich & Markovitch [9], Anderka & Stein [2] applied the ESA method not to Wikipedia but to the Reuters corpus of newswire articles and got similar results, claiming that ESA is independent of the corpus.

14.7 Skip-gram Embeddings

14.7.1 Theory

A *skip-gram* is not an Australian kind of n-gram, but an n-gram in which we allow gaps. Formally, following Guthrie *et al.* [10], a k-skip-n-gram in a sequence of tokens is a sub-sequence (same order of tokens) of n tokens with a maximum of k gaps. In the following, we will deal with 1-skip-n-grams, where we allow a single gap at most.

The *skip-gram model* [10] is a method to obtain a vector representation of words called an *embedding*. The idea is to start with a random vector representation and then improve it in small steps, making it more efficient. How do we measure its efficiency? Every word w splits the total set of words (called the *vocabulary*) into two classes: those that can be part of w's context and those that cannot be part of w's context. It is easy to find examples: "quasar" (a term used in astronomy) probably never occurs in the context of "crochet" (fabric-creating craft). Of course, one can always try to find a sentence using two specific words. When we asked GPT to write a sentence using the two words, it produced

> Although Jane loves to study quasars and other celestial objects, she also enjoys spending her free time practicing crochet to make beautiful blankets and scarves. (ChatGPT)

No matter the words we choose, there can always be a sentence using them that can be imagined and hence written (be it a sentence of the kind "quasars and crochets have nothing in common," which, moreover, is true), but for it to attain statistical significance, it must be part of some communication goal of some people, and this narrows down the breadth of word contexts. An embedding is built out of a corpus of sentences, preferably a large one. By counting the number of times word c is in the context of word w (the context being a window of fixed width), we estimate the probability $P(c \in C(w))$ (where C means "context") of this event. To obtain vectors for w and c, we need some connection between the probability and the vectors as an ultimate goal.

This connection is the following: if $\mathbf{c}$ and $\mathbf{w}$ are the vectors assigned to c and w, we want to have the following equality:

$$P(c \in C(w)) = \frac{1}{1 + e^{-\mathbf{c}\cdot\mathbf{w}}} = \sigma(\mathbf{c} \cdot \mathbf{w}),$$

where σ is the *sigmoid* function. We will make some assumptions for simplification: we will consider that the events $c \in C(w)$ and $c' \in C(w)$ for $c \neq c'$ are independent. Therefore, the probability of having a specific context is the product of the probabilities of having each c_i in the context of w:

$$P(\{c_1, \ldots, c_n\} = C(w)) = \prod_{i=1}^{n} P(c_i \in C(w)) = \prod_{i=1}^{n} \sigma(\mathbf{c}_i \cdot \mathbf{w}).$$

Similarly, the probability of none among $\{c_1, \ldots, c_k\}$ being in the context of w is

$$P(\{c_1, \ldots, c_k\} \cap C(w) = \emptyset) = \prod_{i=1}^{k} P(c_i \notin C(w)) = \prod_{i=1}^{k} \sigma(-\mathbf{c}_i \cdot \mathbf{w}).$$

This formula is important because the skip-gram method greatly emphasizes that many words exist, but few are chosen as members of the context. Therefore, the learning process will use more *negative* samples (words not in the target word's context) than positive samples. We will use the rule that for every observed word c in the context of w, we will introduce K randomly selected words (hoping that we will not happen to choose a word that is, in fact, in the context of w). The width of the sliding window and the number of negative samples per positive sample are the two main parameters of the skip-gram method.

Let us now write the *loss function* that we want to minimize. For every c in the context of w, we randomly select $c'_1, \ldots, c'_K$ that are supposed not to be in the context of w. The probability of this event, taking into account that we consider these events to be independent, will be

$$P(c \in C(w)) \prod_{i=1}^{K} P(c'_i \notin C(w)) = \sigma(\mathbf{c} \cdot \mathbf{w}) \prod_{i=1}^{K} \sigma(-\mathbf{c}'_i \cdot \mathbf{w}).$$

We want to maximize this probability, which is multiplicative and, therefore, involves very small values. We will turn it into an additive property by applying a log function and multiplying by (-1) so that it becomes a minimization function, our *loss function*:

$$L = -[\log\sigma(\mathbf{c}\cdot\mathbf{w}) + \sum_{i=1}^{K}\log\sigma(-\mathbf{c}'_i\cdot\mathbf{w})].$$

Remember that our goal is to start with random values of the vectors and to improve them iteratively until we get close enough to our goal. Modifying vectors is something the computer can do easily, but in what direction?

This is where a popular optimization method comes into play: *gradient descent*. The idea is as follows: imagine you have a function f of many variables $x_1, \ldots, x_n$ in a multi-dimensional space, and you want to find a minimum of f, be it a local minimum. You start with an arbitrary value of the variables and calculate the partial derivatives of the function at that value. If we take each variable x_i separately, then $x_i \mapsto f(x_1, \ldots, x_i, \ldots, x_n)$ is a function of this variable, and $\partial f/\partial x_i$ is the value of its growth for an infinitesimal increase of x_i, therefore if we move x_i in the opposite direction as $\partial f/\partial x_i$, we will contribute in decreasing the value of f. We will use a step size η and replace $\mathbf{x}$ by $\mathbf{x} - \eta(\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$. Once this is done, we recalculate the partial derivatives and move again in the opposite direction.

In our case, L has $K+2$ parameters: w, c and the set of c'_i, for $1 \leq i \leq K$. We consider each one of them as a variable and calculate the partial derivatives:

$$\begin{aligned}
\frac{\partial L}{\partial w} &= (\sigma(\mathbf{c}\cdot\mathbf{w}) - 1)\mathbf{c} + \sum_{i=1}^{K}\sigma(\mathbf{c}'_i\cdot\mathbf{w})\mathbf{c}'_i,\\
\frac{\partial L}{\partial c} &= (\sigma(\mathbf{c}\cdot\mathbf{w}) - 1)\mathbf{w},\\
\frac{\partial L}{\partial c'_i} &= \sigma(\mathbf{c}'_i\cdot\mathbf{w})\mathbf{w}.
\end{aligned}$$

If we update our parameters as follows:

$$\begin{aligned}
w &\leftarrow w - \eta(\sigma(\mathbf{c}\cdot\mathbf{w}) - 1)\mathbf{c} + \sum_{i=1}^{K}\sigma(\mathbf{c}'_i\cdot\mathbf{w})\mathbf{c}'_i,\\
c &\leftarrow c - \eta(\sigma(\mathbf{c}\cdot\mathbf{w}) - 1)\mathbf{w},\\
c'_i &\leftarrow c'_i - \eta\sigma(\mathbf{c}'_i\cdot\mathbf{w})\mathbf{w},
\end{aligned}$$

then we make a step in the right direction. But we cannot spend more time on that specific word. We will use a special kind of gradient descent, called *stochastic descent*, which examines a word at a time, makes the update described above, and then moves on to the next word.

Once we are finished going through the corpus, we use the vectors we obtained as rows of matrices W and C for targets and contexts. Out of these, we build an embedding by summing up the vectors or using only W.

14.7.2 Word Analogies

Embeddings can measure word relatedness through the *cosine similarity* measure of their vectors. But they have another attractive property, known as *word analogy*. The idea is that every kind of relationship between words corresponds to a vector in the high-dimensional space of the embedding. For example, there is a vector for going from singular to plural, a vector for going from female to male, etc. We will test this feature by building an embedding out of the Whovian corpus and comparing it with a pre-existing embedding called *GloVe*.

Here is the code for generating an embedding using the *gensim* package:

```
1  from gensim.test.utils import datapath
2  from gensim import utils
3  import gensim.models
4
5  class MyCorpus:
6      """An iterator that yields sentences (lists of str)."""
7      def __iter__(self):
8          corpus_path = datapath('<absolute path>/dr-who.txt')
9          for line in open(corpus_path):
10             yield utils.simple_preprocess(line)
11
12 sentences = MyCorpus()
13 model = gensim.models.Word2Vec(sentences=sentences)
14 model.save("dr-who.model")
```

where on line 8, we need the absolute path to the corpus file. This file should contain one line per sentence and no punctuation (words separated by whitespace). On line 14, we save the binary model into file `dr-who.model` (around 11MB for a raw text file of 13.5MB). This is a very small model compared to the ones based on multi-gigabyte corpora, but we can still check whether there is some word analogy.

We will apply the "plural → singular" vector to "women" and to "daleks" (expecting to find "woman" and "dalek") and the "male → female" vector to "boy" (expecting to find "girl"). As our embedding space is vast, we cannot expect exact matches. Instead of an exact match, we will look at the words, the vectors of which are geometrically closest to the coordinates we will find. Here is the code:

```
from gensim.models import Word2Vec

model = Word2Vec.load("dr-who.model")

men = model.wv['men']
man = model.wv['man']
women = model.wv['women']
woman = model.wv['woman']
boy = model.wv['boy']
daleks = model.wv['daleks']
print(model.wv.similar_by_vector(women + (man - men), topn=10))
print(model.wv.similar_by_vector(daleks + (man - men), topn=10))
print(model.wv.similar_by_vector(boy + (woman - man), topn=10))
```

On line 11, we ask for the word-vectors that are closest to “women” plus the difference between “man” and “men” (this difference is a vector going from plural to singular; we could have used other word pairs, such as “house” and “houses,” but as “man” and “men” are relatively frequent words this word pair may give more precise results). Here are the results of the “plural → singular” test, applied to “women”:

man (0.73), woman (0.67), girl (0.56), gentleman (0.53), bloke (0.52), guy (0.52), person (0.51), scientist (0.49), chap (0.47), soldier (0.44)

We notice that “woman” does appear, not in the first but in the second position, but this is already extraordinary considering the small size of the embedding. We also notice that all words apply to humans, so we are in the right semantic domain. The same test applied to “daleks” failed miserably:

man (0.66), creature (0.63), daleks (0.59), doctor (0.57), truth (0.56), master (0.53), war (0.53), woman (0.52), person (0.52), mara (0.51)

We do find “creature” and “mara” (a nefarious snake-like entity from the 19th and 20th seasons). Still, otherwise, the embedding didn’t capture the fact that “dalek” is the singular of “daleks” (which is obvious to humans, remember the “wug test” on p. 80). Our embedding failed to capture the “male → female” relationship between “boy” and “girl.” As the female mapping of “boy” we got:

boy (0.70), dear (0.67), mummy (0.61), child (0.60), ryan (0.57), mum (0.56), tegan (0.55), darling (0.55), goodbye (0.53), sarah (0.53)

In the embeddings’ defense, one could observe that “mummy,” “mum,” “tegan,” and “sarah” (the latter are two of the Doctor’s female companions) are referring to female entities. However, the vector of “girl” still does not appear among the ten closest vectors even though the word “girl” appears no less than 1,116 times in the corpus.

Let us compare our mini-embedding with the real thing. “GloVe” stands for “Global Vectors for Word Representation” [19]. It is a family of freely distributed embeddings of various sizes and various origins, among which there are four based on the 2014 English Wikipedia and the Gigaword 5 corpus, a corpus based on 2009–2010 newswire[7]. The training corpus contained 6 billion tokens and a vocabulary of 400k words (by comparison, our mini-embedding was based on 2.5M tokens and a vocabulary of 31k words).

GloVe is very popular so that it can be accessed simply through the *gensim* package. Here is the code for performing the same tests as above using the GloVe embedding (we took the second-smallest GloVe version, using an embedding space of dimension 100):

```
import gensim.downloader as api
model = api.load("glove-wiki-gigaword-100")

men = model['men']
man = model['man']
women = model['women']
```

[7] https://catalog.ldc.upenn.edu/LDC2011T07

```
woman = model['woman']
boy = model['boy']
daleks = model['daleks']
print(model.similar_by_vector(women + (man - men), topn=10))
print(model.similar_by_vector(daleks + (man - men), topn=10))
print(model.similar_by_vector(boy + (woman - man), topn=10))
```

The results are spectacular: when looking for the singular of "women," we get

> woman (0.88), man (0.84), child (0.76), life (0.76), girl (0.74), young (0.74), mother (0.74), she (0.73), one (0.72), women (0.72)

and when looking for the feminine of "boy," we get

> girl (0.91), boy (0.85), woman (0.84), mother (0.77), child (0.74), pregnant (0.73), girls (0.72), baby (0.72), toddler (0.71), daughter (0.70)

It may strike the reader that the second-ranked answer is "boy," the word we started with. This illustrates that there is no "logic" behind this mapping, as would be a query to a knowledge base, but only geometric proximity. The vectors for "boy" and "girl" are close to each other (there are many sentences where the two words are near each other), so that when we find "girl," no matter what the operation that led us, there, we also find "boy" as one of the ten closest vectors.

The reader may wonder how a "generic" embedding such as GloVe reacted to the query on the singular of "daleks". Here are the results:

> daleks (0.75), davros (0.61), tardis (0.57), dalek (0.55), villain (0.54), nemesis (0.53), incarnation (0.51), magneto (0.51), voldemort (0.51), werewolf (0.51),

The correct answer is found, but only at the 4th position. Interestingly, the first six results are very Whovian, while the last three belong to other fictional domains: Marvel for "magneto," Harry Potter for "voldemort," and European folklore for "werewolf." And they are all "villain" figures, so one may argue that GloVe has captured the villainous nature of the Daleks, and this is extraordinary when we consider that the embedding was trained not on fan blogs or social media, but on Wikipedia and official newswire articles. See Exercise 13-3 on p. 423 for a more thorough investigation of GloVe's properties.

14.7.3 Visualization

There is a very nice online tool for visualizing embeddings called *Project Tensorflow Embedding Projector*[8] [21]. All is needed is to upload two files, one containing the vector coordinates (a vector per line) and another containing the words (a word per line in the same order as the vectors in the other file). As we are dealing with a different package (*tensorflow*) than the one of the previous section (*gensim*), here is how to convert the model we created into a format compatible with the Embedding Projector:

---

[8] https://projector.tensorflow.org/

```
1  import io,re
2  from gensim.models import Word2Vec
3  from gensim.models import KeyedVectors
4
5  model = Word2Vec.load("dr-who.model")
6  model.wv.save_word2vec_format('vectors.kv', binary=False)
7
8  f=io.open("vectors.kv",mode='r',encoding='utf-8')
9  o1=io.open("vectors.tsv",mode='w',encoding='utf-8')
10 o2=io.open("metadata.tsv",mode='w',encoding='utf-8')
11 for l in f:
12     if (re.match(r'^[0-9]+ [0-9]+',l)):
13         pass
14     elif (re.match(r'^([^ ]+) ([0-9e. -]+)',l)):
15         m=re.match(r'^([^ ]+) ([0-9e. -]+)',l)
16         print("\t".join(m.group(2).split()),file=o1)
17         print(m.group(1),file=o2)
18 f.close()
19 o1.close()
20 o2.close()
```

The test on line 12 prevents the file header (containing the number of vectors and the dimension of the embedding space) from being considered a vector. The `e` in the regular expressions of lines 14 and 15 is crucial because when coordinates get very small, scientific notation is used so that values such as `7.734467e-05` can be found in the data. Not reading these results in data corruption.

Once uploaded via the "Load" button on the Embedding Projector's page, we see a dynamic 3D representation of the embedding, as in Fig. 14.2. In the right part of the window, one can choose a word, and then the word and its closest neighbors are highlighted. The 3D representation of the embedding is dynamic: one can turn, shift, and zoom it freely.

14.7.4 FastText

One of the main problems of word-level embeddings is the missing word problem: when a word does not occur in the corpus because it is a neologism or a very rare word or because it contains some spelling error, the skip-gram model cannot process it. Of course, when searching in the GloVe embedding for the neighbors of "color," we do get some variants

colour (0.87), colors (0.82), colored (0.71), colours (0.65), etc.

where "colour" is the UK spelling of "color." Sometimes one can even detect spelling errors in embeddings. For example, there is indeed an entry for *"collor" in GloVe. But this can only be a lucky break. One can imagine other errors, such as *"cholor," or *"kolor," or *"colorr," which do not appear in GloVe.

To solve this problem, the `fasttext` encoding [5] uses, besides entire words, graphemic subsequences. For example, in a configuration where graphemic trigrams

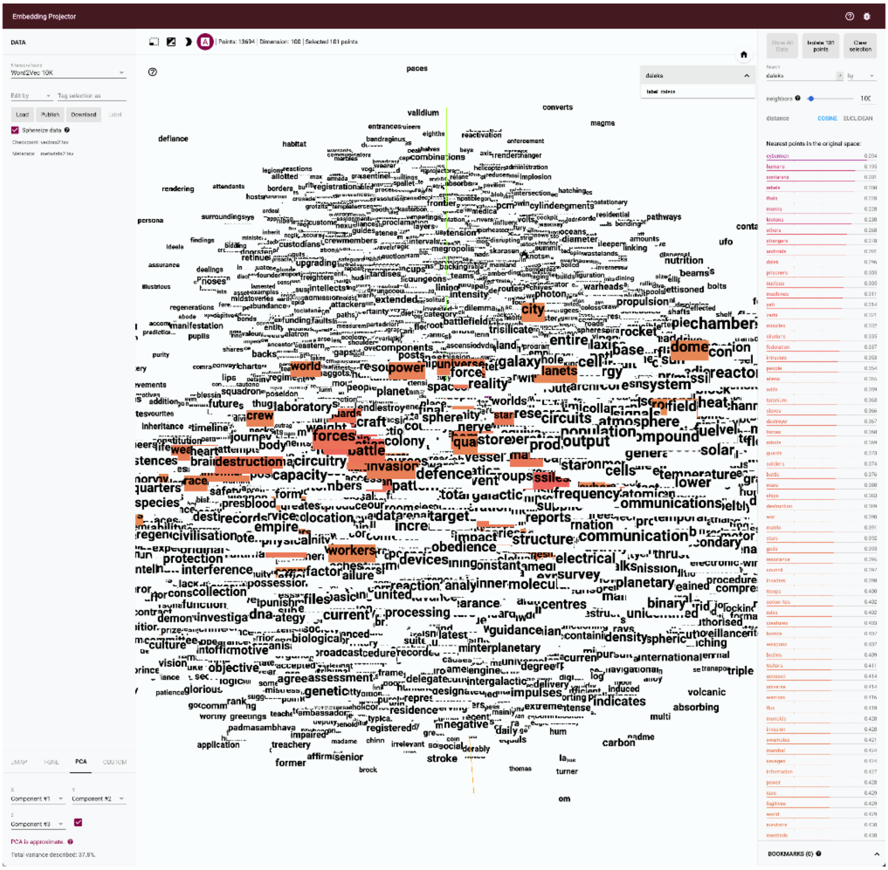


Fig. 14.2 Visualizing the Whovian corpus embedding through the Project Tensorflow Embedding Projector tool

are used, the word "Tardis" will produce a token `<tardis>` (where < and > are word boundary symbols) as well as tokens `<ta`, `tar`, `ard`, `rdi`, `dis`, `is>`. The embedding space is then structured by the vector representations of the graphemic trigrams, and a word vector is stored as the sum of the vectors of its graphemic trigrams.

When querying the representation of a word that is not in the vocabulary of the corpus, this word is broken into graphemic trigrams, and at least some of them almost surely will be represented in the embedding space. Then, the unknown word will be represented by the sum of the vectors of the recognized graphemic trigrams. To evaluate the chances that a given trigram appears in the corpus, we have calculated trigrams out of all purely alphabetic words of the Whovian corpus (words that contain only letters and potentially an apostrophe), and we have found 7,235 such graphemic trigrams. The theoretic total number of such trigrams is $26^3+3\times26^2+2\times26^2+4\times26$ (patterns `xxx`, `'xx`, `x'x`, `xx'`, `<xx`, `xx>`, `<'x`, `<x'`, `'x>`, `x'>`, where `x` is any letter), that is 21,060. So, theoretically, we have a 0.3435 probability that a given graphemic trigram has occurred in the corpus. This probability is much higher because the graphemic trigrams must obey the graphotactic rules of the language. To evaluate the size of the space of graphemically allowed trigrams, we took a dictionary of 40,926 English lemmas (we can reasonably assume that the inflected versions of these lemmas will not add many trigrams to the uninflected ones) and calculated the number of distinct trigrams; it is 7,770. In other words, the Whovian corpus already covers 93% of the total number of trigrams needed for English words, which is quite sufficient.

The code for using FastText in the *gensim* package is similar to the one we saw previously. First, we create the model:

```
from gensim.models import FastText
from gensim.test.utils import datapath
from gensim import utils

class MyCorpus:
    """An iterator that yields sentences (lists of str)."""
    def __iter__(self):
        corpus_path = datapath('<absolute path>/dr-who-word2vec.txt')
        for line in open(corpus_path):
            yield utils.simple_preprocess(line)

import gensim.models

sentences = MyCorpus()
model = FastText(sentences=sentences)
model.save("dr-who-fasttext.model")
```

and then we use it, for example, to find words, the vectors of which are close to vectors of words not appearing in the corpus:

```
from gensim.models import FastText
model = FastText.load("dr-who-fasttext.model")
print(model.wv.most_similar('tardislike', topn=10))
print(model.wv.most_similar('dalekish', topn=10))
print(model.wv.most_similar('dona', topn=10))
```

The first word, "tardislike," may not occur in the Whovian corpus but does have an entry in the English Wiktionary. Here are the results we obtained:

> tardis (0.85), tardises (0.84), tar (0.83), jeopardise (0.82), cardiff (0.81), rouse (0.80), cowardice (0.79), motorbike (0.79), wheelhouse (0.79), shine (0.78)

the two first candidates are very relevant. The fifth one is interesting: Cardiff is the place where the Dr. Who Museum[9] was located until it closed in 2017, and some episodes were played there.

The word "dalekish," is used in the comment of a Whovian blog,[10] but otherwise not in the Whovian corpus. It gives

> dalek (0.93), daleks (0.93), dalekanium (0.92), dale (0.91), dalios (0.89), cyberiad (0.88), cybermen (0.87), malek (0.87), cyberium (0.86), cyberman (0.86)

---

[9] https://en.wikipedia.org/wiki/Doctor_Who_exhibitions

[10] https://perma.cc/R7MD-E3RP

which is impressive: not only do we get the singular and plural of "Dalek," but also the "Dalekanium" and four terms related to Cybermen, the vectors of which seem to be close to the vectors of Daleks. And finally, the spelling error *"Dona." The Doctor's companion's name is "Donna" (with two 'n')—there are five occurrences of "Donald" (including one for Donald Trump) in the corpus and one word "Donated," but never any "Dona". Here are the results:

donna (0.94), donald (0.91), donor (0.90), don (0.89), doña (0.89), dorf (0.88), doh (0.87), dot (0.86), dolt (0.84), dodo (0.83)

The correct "Donna" is first with high cosine similarity, then "Donald" and other words starting with "Do-." As we see, FastText works quite well for English, and it should work correctly for most alphabetic writing systems. It would be interesting to test it for Arabic and Chinese: for the former, there is a chance it may work by (incidentally) separating affixes from roots, but for the latter, its utility is probably limited.

14.8 Further Reading

14.8.1 Literature

The undisputed Bible of statistical methods in NLP is Manning & Schütze [17]. The book was released in 1999, but it is still a very insightful read, and for many, it has been the primary source from which they learned statistical NLP. There is also a companion website,[11] last updated in 2015. Also from Stanford and rich in information, but different in its topics and approach is Jurafsky & Martin [13]. While Manning & Schütze was a one-shot, Jurafsky & Martin is already in its third edition. The changes between editions are far from just cosmetic details: the book keeps track of the field's evolution and changes considerably from one edition to another. The 3rd edition is still under preparation, and all chapter drafts are available on the Web,[12] including some appendices that will be on the Web only.

Besides these two Leviathans of statistical NLP, we can also mention Bird [4], which is dedicated to the *nltk* Python package but is far more than a user's manual since it explains the various NLP topics (neither Manning & Schütze nor Jurafsky & Martin include any code). Once again, there is a Web site[13] with an updated version of Bird [4], compatible with Python 3 and the latest version of *nltk*.

Eisenstein [8] appeared more recently. It is a pretty dense text with far more than statistical NLP.

As for the history of NLP, and statistical NLP in particular, Léon [16] is a fascinating read.

[11] https://nlp.stanford.edu/fsnlp/

[12] https://web.stanford.edu/~jurafsky/slp3/

[13] https://www.nltk.org/book/

14.9 Exercises

Exercise 13-1: Exploring the Anthology of the Association for Computational Linguistics

The *Association for Computational Linguistics* (ACL) is the top scientific and professional organization for those working in Natural Language Processing. Its journals and conferences are the best in the world, and all papers are freely available on the Association's Web site.[14]

Find the most central ACL author for each decade of the 20th and 21st centuries (starting with the fifties).

Find the big families of topics to which the 88,065 (as of October 2023) papers belong.

Exercise 13-2: Find a Cornelian play among nine Racinian plays

Racine (1639–1699) and Corneille (1606–1684) were the two most important French tragedians of the 17th century (Molière, who is even more popular, mostly wrote comedies). We have prepared a corpus of ten plays, from which we have removed all metadata (speakers, stage instructions, etc.). Nine of these plays are written by Racine and one by Corneille. Find the Cornelian play.

Exercise 13-3: Test properties of GloVe embeddings

Use hyperspherical coordinates to evaluate the gender, hyperonymy, and antonymy properties of GloVe.

References

[1] John R. Allen. “Methods of Author Identification through Stylistic Analysis”. In: *The French Review* 47.5 (1974), pp. 904–916.

[2] Maik Anderka and Benno Stein. “The ESA retrieval model revisited”. In: *Proceedings of the 32nd International ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 2009, pp. 670–671.

[3] Jean-Pierre Benzécri et al. *Pratique de l’analyse des données 3. Linguistique & Lexicologie*. Dunod, 1981.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O’Reilly, 2009.

[5] Piotr Bojanowski et al. “Enriching word vectors with subword information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

---

[14] https://aclanthology.org/

[6] Mark Davis and Christopher Chapman. *Unicode Text Segmentation* (originally called *Text Boundaries*). Vol. 29. Unicode Standard Annexes. https://www.unicode.org/reports/tr29/. 2001–2022.

[7] Marie-Louise Dufrénoy. "Analyse statistique du langage". In: *Journal de la Société de statistique de Paris* 87 (1946), pp. 208–219.

[8] Jacob Eisenstein. *Introduction to Natural Language Processing*. The MIT Press, 2019.

[9] Evgeniy Gabrilovich and Shaul Markovitch. "Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis". In: *Proc. of IJCAI-07*. 2007, pp. 1606–1611.

[10] David Guthrie et al. "A Closer Look at Skip-gram Modelling". In: *Proceedings of LREC 2006*. 2006, pp. 1222–1225.

[11] Yannis Haralambous and Vitaly Kluyev. "A semantic relatedness measure based on combined encyclopedic, ontological and collocational knowledge". In: *International Joint Conference on Natural Language Processing, Chiang Mai*. 2011.

[12] Zellig S. Harris. "Distributional structure". In: *Word* 10 (1954), pp. 146–162.

[13] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Third Edition Draft, https://web.stanford.edu/~jurafsky/slp3/. 2023.

[14] Casper J. Labuschagne. *Numerical Secrets of the Bible. Rediscovering the Bible Codes*. North Richland Hills, TX: Bibal Press, 2000.

[15] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. "Introduction to Latent Semantic Analysis". In: *Discourse Processes* 25 (1998), pp. 259–284.

[16] Jacqueline Léon. *Automating Linguistics*. Springer, 2021.

[17] Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[18] Andrey A. Markov. "An Example of Statistical Investigation of the Text Eugene Onegin Concerning the Connection of Samples in Chains. (Lecture at the physical-mathematical faculty, Royal Academy of Sciences, St. Petersburg, 23 January 1913)". In: *Science in Context* 19 (2006), pp. 591–600.

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[20] Pomaret. "L'Architecture du langage". In: *Revue Générale des Sciences Pures et Appliquées* 50 (1939). 51 (1940), 70–75, pp. 152–159, 408–412.

[21] Daniel Smilkov et al. "Embedding Projector: Interactive Visualization and Interpretation of Embeddings". In: *NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems*. 2016.

[22] Kam-Fai Wong et al. *Introduction to Chinese Natural Language Processing*. Vol. 4. Synthesis Lectures on Human Language Technologies. Morgan & Claypool, 2010.

Chapter 15
Going Neural

> We, deep neural networks, are great at recognizing cats and dogs. But sometimes we get confused and end up labeling a horse as a "long-necked dog" or a giraffe as a "spotted horse." It's a good thing we're not in charge of naming species!
>
> ChatGPT, May 16, 2023

15.1 Feedforward Neural Networks

A *feedforward deep neural network* consists of

1. a certain amount of *layers* containing a (large) amount of *neurons* or *nodes*. A node is a small cell with a series of parameters (as many as neurons in the previous layer, plus one) and a value. The neuron j of a given layer $[\ell]$,
    - receives values from all cells of the previous layer $[\ell - 1]$ (or the outside world if it is the first layer),
    - multiplies each value received from neuron i of level $[\ell - 1]$ by a parameter value $w_{i,j}^{[\ell]}$,
    - adds a value $b_j^{[\ell]}$ to it, called *bias*,
    - applies a (preferably non-linear) function to it, called *activation function* (this can be the sigmoid function or the hyperbolic tangent tanh, or the ReLU function (constant 0 for negative values and $x \mapsto x$ for positive values),
    - sends the value calculated to all neurons of the next layer (or to the outside world if it is the last layer);
2. a *loss fuction* L that evaluates the difference between the result, output by the last layer of the network, and the expected result;
3. an algorithm for updating the values of parameters of all nodes, layer by layer, starting with the last layer and going backward until the first layer. If this algorithm is gradient descent, then we need a *learning rate* η;

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_15

4. potentially a special output layer that will normalize the results received from the previous layer. One of the most popular functions used for normalization is *softmax*:

$$\mathsf{softmax}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} := \frac{1}{\sum e^{x_i}} \begin{pmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{pmatrix}.$$

   As the reader can verify, softmax "cleans up all the mess" of the values received and returns a value between 0 and 1, which can be assimilated to a probability.

A deep neural network can have millions of parameters. Where do they get their values from? Here is the trick: it starts with random values, calculates a first result starting from an observation (called *propagation*), and compares the result obtained with the expected result. It calculates the error and applies gradient descent to modify the values of the parameters (this is called *backpropagation*). Then it starts again: propagation, backpropagation, propagation, backpropagation, etc. A propagation/backpropagation cycle is called an *iteration*. After doing this cycle hundreds of times, either the whole set of parameters goes wild or converges to the appropriate values for doing the task the network is intended for. The saying goes, "You can't teach an old dog new tricks," where "You" refers to humans. Well, machines are not humans, and neural networks are not old dogs so that they can learn "new tricks" and do it all the time.

15.1.1 Doing It Manually

This section will give an example of a toy deep neural network with only three layers and five nodes. It has been adapted from Skansi [15, p. 95], who, in turn, adapted it from a blog[1]. To get an intuition of how neural networks work, we will manually do the calculations of an iteration. It will be no picnic, but by going through it, the reader will be satisfied with mastering what happens inside the blackest among the black boxes of machine learning.

The network can be seen in Fig. 15.1. It consists of five nodes, called A, B, C, D, and E, storing values x_A, x_B (x for the first layer, i.e., the input values), y_C, y_D (y for the second layer) and z_E, which is also the output value. We consider that the activation function is the sigmoid σ. The parameters are w_i, as displayed in the figure, next to the network's edges. To simplify, we consider that there is no bias. We consider that in our training set, we have only one individual, namely $(0.23, 0.82)$, so that the values of the first layer are fixed: $x_A = 0.23$ and $x_B = 0.82$ and will remain so. We will consider that the correct output value for this individual is 1.

The initial parameter values are (supposedly) random:

[1] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

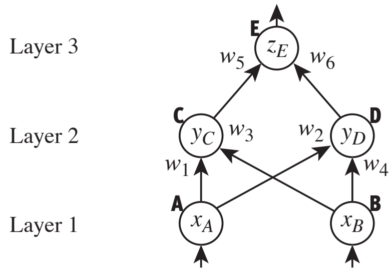


Fig. 15.1 A toy deep feedforward neural network, adapted from [15, p. 97]

w_1	w_2	w_3	w_4	w_5	w_6
0.1	0.5	0.4	0.3	0.2	0.6

We said that every node gathers values from the nodes of the previous layer, multiplies them by the appropriate w_* parameter, takes the sum of these values, and applies the activation function (in our case, the sigmoid σ) to it. For example, what arrives to C is the values of x_A and x_B, the former multiplied by w_1 and the latter by w_3. Once the calculation $w_1x_A + w_3x_b$ done (which is actually the scalar product of vectors (w_1, w_3) and (x_A, x_B)) the activation function σ is applied to it, the value of cell C becomes $y_C = \sigma(w_1x_A + w_3x_B)$, which is also the value that *leaves* C heading towards E on the third layer.

Therefore the value y_C that *leaves* node C will be $y_C = \sigma(w_1x_A + w_3x_B)$.

Therefore, the first propagation is as follows:

$$\begin{aligned}
y_C &= \sigma(w_1x_A + w_3x_B) = \sigma(0.23 \cdot 0.1 + 0.82 \cdot 0.4) = 0.5868,\\
y_D &= \sigma(w_2x_A + w_4x_B) = \sigma(0.23 \cdot 0.5 + 0.82 \cdot 0.3) = 0.5982,\\
z_E &= \sigma(w_5y_C + w_6y_D) = \sigma(0.5868 \cdot 0.2 + 0.5892 \cdot 0.6) = 0.6155.
\end{aligned}$$

Let us take a quadratic loss function $L = \frac{1}{2}(z_E - f(x_A, x_B))^2$, where f is the function we want to approximate with our network. In our case $f(x_A, x_B) = 1$ and therefore the loss will be

$$L = \frac{1}{2}(0.6155 - 1)^2 = 0.0739.$$

The gradient descent algorithm, as we saw in the previous chapter, consists of taking the partial derivative of the loss function for each one of its parameters and changing the values of the parameters by subtracting from them η times the partial derivative of the loss function, where η is the learning rate.

We now have some heavy partial derivative calculations to do. Let us do them in detail for w_5, applying the general rule $(f \circ g)'(x) = f'(g(x))g'(x)$:

$$\frac{\partial L}{\partial w_5} = \frac{\partial}{\partial w_5}\left(\frac{1}{2}(z_E - 1)^2\right)$$

$$
\begin{aligned}
&= \frac{\partial}{\partial w_5}\left(\frac{1}{2}(\sigma(w_5 y_C + w_6 y_D) - 1)^2\right)\\
&= (\sigma(w_5 y_C + w_6 y_D) - 1)\frac{\partial}{\partial w_5}\sigma(w_5 y_C + w_6 y_D)\\
&= (\sigma(w_5 y_C + w_6 y_D) - 1)\sigma(w_5 y_C + w_6 y_D)(1 - \sigma(w_5 y_C + w_6 y_D))\frac{\partial}{\partial w_5}(w_5 y_c)\\
&= (\sigma(w_5 y_C + w_6 y_D) - 1)\sigma(w_5 y_C + w_6 y_D)(1 - \sigma(w_5 y_C + w_6 y_D))y_c\\
&= (0.6155 - 1) \times 0.6155 \times (1 - 0.6155) \times 0.5868\\
&= -0.0534.
\end{aligned}
$$

Considering that we take an learning rate $\eta = 0.7$, our new $w_{5,\mathsf{new}}$ will be

$$w_{5,\mathsf{new}} := w_5 - \eta\frac{\partial L}{\partial w_5} = 0.2 - 0.7 \times (-0.0534) = 0.23738.$$

In other words, when we will propagate anew, w_5 will be equal to 0.23738, that is an increase by 19%.

We do the same calculations for the other variables. To take the partial derivative of L with respect to, for example, w_1, it suffices to replace y_C by its formula in the formula of z_E:

$$z_E = \sigma(w_5 y_C + w_6 y_D) = \sigma(w_5\sigma(w_1 x_A + w_3 x_B) + w_6 y_D).$$

After completing all the tedious calculations, we get the following table:

	w_1	w_2	w_3	w_4	w_5	w_6
old values	0.1	0.5	0.4	0.3	0.2	0.6
new values	0.1007	0.502	0.4024	0.307	0.2373	0.6374
change	0.7	0.4%	0.6%	2.3%	18.7%	6.2%

As we see, the parameters closer to the output layer have undergone more significant changes than those closer to the input layer. This is a form of the *vanishing gradient phenomenon*, even though this phenomenon becomes much more important in the case of recurrent neural networks (§15.2).

Normally, the network moves on to the next individual once the iteration cycle is over. In our toy example, we can start the propagation anew on the same individual with the new parameter values. The output value will be 0.6261, and therefore, a new loss value of

$$L_{\mathsf{new}} = \frac{1}{2}(1 - 0.6261)^2 = 0.0699,$$

i.e., an improvement of 5.5%. This is not much, but after all, we have only one individual, and we ran the iteration cycle only once.

15.1.2 Batches and Epochs

When you have a million individuals to propagate to train the model, not only would it be time-consuming to backpropagate after each individual, but it would also cause useless drastic changes to the parameters on every outlier or every wrongly labeled individual. Therefore, it is more convenient to propagate *batches* of individuals and only backpropagate at the end of each batch. The batch size is an *hyperparameter* of the system, i.e., a parameter set by the human operator of the system and not a parameter learned from the data.

Furthermore, believing that a single run through all batches will bring the expected result would be illusory. As ancient Greeks and the well-known educator Zig Ziglar (who is neither Greek nor ancient) used to say, “Repetition is the mother of learning,” and this also goes for machine learning. To learn, one has to repeat the operation and read the same individuals again and again until the accuracy (or F1-measure) of the result is satisfactory (provided the system is converging). Each time we process the entire training set, we call it an *epoch*.

15.1.3 How To Read Graphical Illustrations of Neural Networks

When studying neural networks, at the beginning, they are complex. Later, they become *very* complex, or *even very* very complex, both in their internal workings, the flow of information transmitted, and all their bells and whistles. Any decent textbook on neural networks uses graphical illustrations to depict them, but due to their complexity, drawing a neural network is almost as tricky as inventing it in the first place. There are several approaches, differing on the level of detail and on the part left to the reader’s imagination to complete.

In Fig. 15.2, we display three types of graphical illustration for the same kind of neural network: a feedforward network with an input layer, two linear layers (of n nodes), and a softmax layer (of m nodes). In Fig. 15.2 (a) nodes are displayed by circles (or boxes with rounded edges) containing the name of a variable. Similarly to Fig. 15.1, we use arrows in the frame of the metaphor of information transmission. Each arrow corresponds to a parameter of the model; there isn’t enough space to include it explicitly in the illustration (as we did in Fig. 15.1), but it is understood that between, for example, node x_i (layer 2) and node y_j (layer 3), there is multiplication by w'_{ij}, which is a cell of the matrix W'. The activation function is not explicitly mentioned either. This illustration shows the complexity of this relatively simple, feedforward neural network. Still, the global set of arrows and not the individual arrows per se contribute to understanding the network.

In Fig. 15.2 (b) a shaded area replaces the mesh of arrows. Jurafsky & Martin also switch from a mesh-based approach (e.g., in [9, Fig. 7.8]) to a mixed one, using mesh and shaded area [9, Fig. 7.13], to a shaded area in [9, Fig. 9.2]. Contrarily to Jurafsky & Martin, we attach the shaded area to the upper layer to show that the corresponding matrix of parameters belongs to the layer above it and not the one

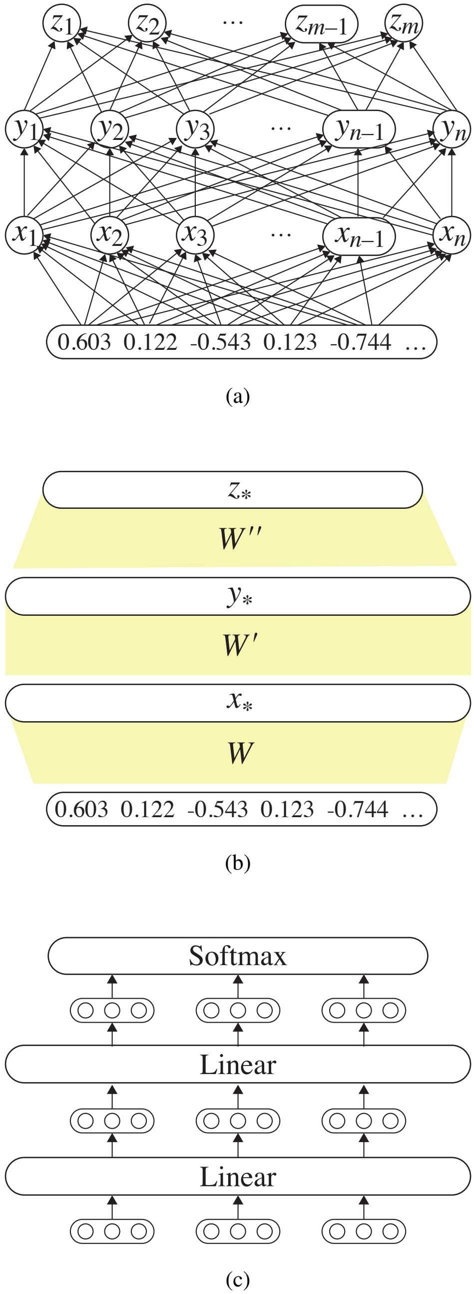


Fig. 15.2 Three different approaches to the graphical illustration of neural networks, inspired by Jurafsky & Martin for (b) [9, Fig. 9.5] and Hagiwara for (c) [7, Fig. 8.10]

below it. Instead of using symbols for each node and each parameter, the former are replaced by a generic symbol and the latter by the symbol of the matrix.

Finally, nodes and parameters are not named in Fig. 15.2 (c). The type of the layer is given instead. The rounded boxes containing circles represent vectors corresponding to elements of the training set (for the lowest level) and their transformations when propagating through the network layers. This graphical approach is well suited for representing recurrent networks (§ 15.2), where small symbols represent unrolled layers. This approach is used by Hagiwara throughout his book [7], with remarkable graphical consistency.

15.1.4 An Example in Python

This section will show an example of implementing text classification using a feedforward neural network. We have collected data from the Web to obtain three subcorpora: the Whovian corpus, which we have used all along this book, a corpus of transcripts of *Star Trek*[2] (the original series) and a corpus of transcripts of the nine *Star Wars* movies[3]. We have gathered speech turns into units of at least 256 characters, keeping complete sentences. The three subcorpora are unfortunately unbalanced: 43,952 units for Dr. Who, 6,805 for Star Trek, and only 1,539 for Star Wars. When we ran the neural network on the unbalanced merge of the three, we got an accuracy of 96%, which is a bit unfair. So we have randomly selected 1,539 units from the two largest subcorpora so that all three subcorpora end up being of equal size.

We used the *keras* package to implement this neural network, and our code is inspired by an example in Chollet [3, pp. 106–112]. Here is the code:

```
import io,re,random
f=io.open("sf.txt",mode="r",encoding="utf-8")
BIG=[]
WORDS={}
COUNTS={}
TAGS={}
TAGcounter=0

for l in f:
    m=re.match(r'([a-z]+)\t(.+)',l)
    tag=m.group(1)
    contenu=m.group(2)
    BIG.append((tag,contenu))
    if tag not in TAGS.keys():
        TAGS[tag]=TAGcounter
        TAGcounter+=1
    if tag not in COUNTS.keys():
        COUNTS[tag]=0
```

[2] http://www.chakoteya.net/StarTrek/

[3] https://imsdb.com/

```
19         COUNTS[tag] += 1
20     f.close()
```

In this part, we read the file, which contains, on each line, a class tag and a text unit, separated by a tabulation. We store the results into `BIG`.

```
21 filteredBIG=[]
22 minTAG=2**32
23 for tag in TAGS:
24     if COUNTS[tag] < minTAG:
25         minTAG = COUNTS[tag]
26         tagMIN = tag
27 for tag in TAGS:
28     tmp=[x for x in BIG if x[0]==tag]
29     random.shuffle(tmp)
30     tmp=tmp[:minTAG]
31     filteredBIG.extend(tmp)
32 for (tag,contenu) in filteredBIG:
33     words=contenu.rstrip().split()
34     for w in words:
35         if w not in WORDS.keys():
36             WORDS[w]=0
37         WORDS[w] += 1
38
39 WORDS={x: WORDS[x] for x in WORDS.keys() if WORDS[x] >= 10}
40 WORDS_SORTED=sorted(WORDS.keys(),key=lambda x: WORDS[x],reverse=True)
41
42 word_counter=2
43 word_index={}
44 word_index['']=0
45 word_index['[UNK]']=1
46 for x in WORDS_SORTED:
47     word_index[x]=word_counter
48     word_counter += 1
```

This is the second part of the data preparation. On lines 23–26, we identify the smallest subcorpus. On lines 27-31 we shuffle each subcorpus separately and keep a part equal to the smallest subcorpus. On lines 32–37, we count words in the filtered corpus so that on line 39 we can keep only those words that occur more than 10 times. We sort words by decreasing frequency. Finally, on lines 42–48, we build our word index, a dictionary mapping words to increasing numbers (according to Chollet's conventions, entry 0 is the empty words, and entry 1 is the tag `[UNK]` for unknown words.

```
49 SENTENCES=[]
50 LABELS=[]
51 random.shuffle(filteredBIG)
52 for (tag,contenu) in filteredBIG:
53     words=contenu.rstrip().split()
54     indexes=[]
55     for w in words:
56         if w in word_index.keys():
57             indexes.append(word_index[w])
58         else:
```

```
59                  indexes.append(1)
60          SENTENCES.append(indexes)
61          LABELS.append(TAGS[tag])
```

We replace words that occur more than 10 times by their indexes and all other words by the index of `[UNK]`, namely `1`. `SENTENCES` is a list of lists of word indices. `LABELS` is a list of numeric labels of the same length as `SENTENCES`.

```
62  TRAIN_SIZE=int(0.7*len(SENTENCES))
63  train_data=SENTENCES[:TRAIN_SIZE]
64  train_labels=LABELS[:TRAIN_SIZE]
65  test_data=SENTENCES[TRAIN_SIZE:]
66  test_labels=LABELS[TRAIN_SIZE:]
```

We separate the corpus into train set (70%) and test set data (30%), together with the appropriate labels.

```
67  import numpy as np
68  def vectorize_sequences(sequences, \
69      dimension=max(10000,len(word_index.keys()))):
70      results = np.zeros((len(sequences), dimension))
71      for i, sequence in enumerate(sequences):
72          for j in sequence:
73              results[i, j] = 1.
74      return results
75
76  x_train = vectorize_sequences(train_data)
77  x_test = vectorize_sequences(test_data)
78
79  from tensorflow.keras.utils import to_categorical
80  y_train = to_categorical(train_labels)
81  y_test = to_categorical(test_labels)
```

The function defined on lines 68–74 will be used to create binary vectors for sequences: each sequence becomes a vector of length equal to the number of existing word indices, it is set to zero, and then, for each word of index k, the k-th entry of the vector is set to 1. On lines 76 and 77, we apply this function to train and test data. On lines 79–81, we convert the list of labels into tensors.

```
82  import keras
83  from keras import layers
84  model = keras.Sequential([
85      layers.Dense(64, activation="relu"),
86      layers.Dense(64, activation="relu"),
87      layers.Dense(len(TAG.keys()), activation="softmax")
88  ])
89
90  model.compile(optimizer="rmsprop",
91                loss="categorical_crossentropy",
92                metrics=["accuracy"])
```

Lines 83–87 show the elegance of the *keras* package: the architecture of the neural model is given by three lines, one for each layer. Only the absolutely necessary information is given: how many neurons are requested and what activation function is chosen. Then, lines 90–92 prepare the ground for the model fitting: we choose the optimizer (i.e., what kind of gradient descent method or similar algorithm allows optimizing parameter values), the loss function, and the metrics we want to monitor. In our case, the optimizer is root mean square propagation, a very popular algorithm. To indicate the learning rate η, it suffices to use `keras.optimizers.RMSprop(learning_rate=1e-4)` instead of `"rmsprop"` (here `1e-4` stands for 1×10^{-4}). We have chosen *categorical cross entropy* as the loss function. It would be *binary cross entropy* if we had only two classes (there are other loss functions, but cross-entropy is the most popular). Finally, we use accuracy to monitor the performance during validation—the reader may wonder why one of the most advanced deep learning packages uses a primitive measure such as accuracy. In fact, precision and recall measures have been removed because they were misleading because data are read in batches.

```
93  history = model.fit(x_train,
94                      y_train,
95                      epochs=20,
96                      batch_size=128,
97                      validation_split=0.3)
```

Now, at last, we fit our model to the data. We request 20 epochs and a batch size of 128 (considering that the train set has 3,231 individuals, a whole epoch requires 26 batches). As for the number of epochs, we will monitor the evolution of accuracy to see whether they were necessary. A validation split of 0.3 means that 30% of the training data will be used for validation.

Validation data is part of the training corpus used during the model fitting process. While we fit the model, we train it on the training data and test it on the validation data after every epoch. As we monitor the advance of the model fit, we see whether validation accuracy is increasing or stagnating. If epoch after epoch, the evolution of validation accuracy is not satisfactory, we can stop the process and try different hyperparameters. Once we have completed the model fitting, we test it on test data. By keeping the three data sets separate, we are sure that the validation and test sentences have not been used for training.

```
98  acc = history.history["accuracy"]
99  val_acc = history.history["val_accuracy"]
100 
101 import matplotlib.pyplot as plt
102 loss = history.history["loss"]
103 val_loss = history.history["val_loss"]
104 epochs = range(1, len(loss) + 1)
105 plt.plot(epochs, acc, "bo", label="Training accuracy")
106 plt.plot(epochs, val_acc, "b", label="Validation accuracy")
107 plt.title("Training and validation accuracy")
108 plt.xlabel("Epochs")
109 plt.ylabel("Accuracy")
110 plt.legend()
111 plt.savefig("sf.pdf")
112 plt.show()
```

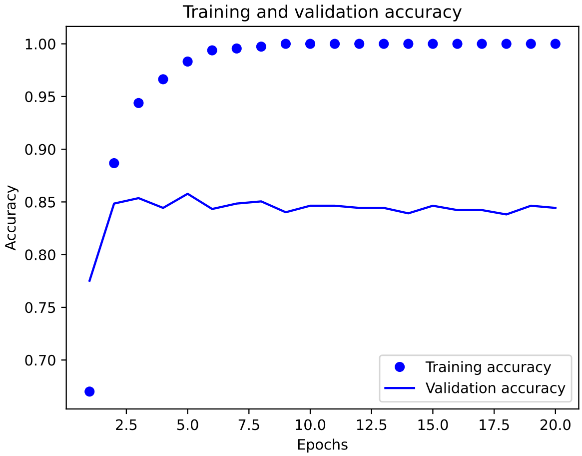


Fig. 15.3 Training and validation accuracies of the feedforward neural network example

This code will display curves for training accuracy and validation accuracy. The diagram obtained can be seen in Fig. 15.3. As we see, training accuracy reaches 100%, while validation accuracy reaches 85% after only four epochs and then stays at that level, with small fluctuations. We can conclude that it was unnecessary to request twenty epochs since we get the best possible performance already after four epochs.

```
113  from sklearn.metrics import classification_report
114
115  y_pred = model.predict(x_test, batch_size=64, verbose=1)
116  y_pred_bool = np.argmax(y_pred, axis=1)
117
118  tags=sorted(TAGS.keys(), key=lambda x:TAGS[x])
119  y_test=np.array([list(x).index(1) for x in y_test])
120
121  print(classification_report(y_test, y_pred_bool,target_names=tags))
```

This final part of the code is the real evaluation of the model. On line 115, we apply the model to the test corpus. The result is a list of lists containing a probability weight for each class. On line 116, we convert this structure into a list of class labels. Lines 118 and 119 do the same for the real class values for the test corpus. Finally, `classification_report` will return a small table containing precision, recall, F1-measure for all classes and accuracy on the global test set.

Here is the output of this method:

```
              precision    recall  f1-score   support

    startrek       0.84      0.84      0.84       462
    starwars       0.90      0.85      0.88       452
       drwho       0.77      0.81      0.79       472
```

```
    accuracy                          0.83      1386
```

As we see, the highest classification performance is attained for *Star Wars*, followed by *Star Trek* and Dr. Who. An accuracy of 83% is quite a satisfying result, considering the small size of our corpus.

Let us try to find out what made this result possible. The first (second, and third) rule when you want to find out why your data have been classified or transformed in some way is to look at the data. Here is a randomly chosen sentence from the corpus (any nonalphabetic characters have been removed):

> i thought i told you when im on the bridge that you wanted the reports by its now sir oh i see thank you shes replacing your former yeoman sir no she does a good job all right its just that i cant get used to having a woman on the bridge

Words like "bridge" and "yeoman" are specific to the *Star Trek* universe (the word "bridge" appears only seven times in the *Star Wars* corpus, "yeoman" not a single time), and then the sexist comment "I can't get used to having a woman on the bridge" points to Captain Kirk rather than the Doctor (even though the "classic" Whovian period has not reached the heights of political correctness either). Another randomly chosen sentence:

> you were just a little boy then it may not be as you remember it time changes your perception i think time has given me much more mature feelings to enhance my perception it must be difficult having sworn your life to the jedi not being able to visit the places you like or do the things you like

The word "jedi" is sufficient to classify this sentence as a *Star Wars* sentence. Otherwise, the choice is difficult: in all three corpora, the main characters occasionally return to memories from their childhood so it could be any of them.

Our first hypothesis is the length of the input strings that favored discrimination between the three subcorpora. In word sequences of this length, there will always be a word that is unique to one of the corpora (like "yeoman" and "jedi" in the two examples above). To check this hypothesis, we have reduced the number of words in each sample and have fitted the model on the reduced samples. Here are the results:

It would be interesting to see to what extent the length of strings is important for the classification. After all, in very long sequences, there is a good chance that some word that allows identifying the corpus is included in the sequence. We have strings with full turns concatenated so that their total length is over 256 characters. Let us see what happens when we reduce this length. We retrain the model by reducing the number of words in each sample (always keeping complete words), and we fit the model on the reduced samples. Here are the results:

	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%
accuracy	0.83	0.83	0.83	0.79	0.75	0.69	0.68	0.64	0.59	0.47
precision Whovian	0.77	0.81	0.77	0.76	0.69	0.64	0.63	0.63	0.55	0.46
recall Whovian	0.81	0.79	0.81	0.70	0.77	0.64	0.62	0.57	0.63	0.49

We can conclude that sentence length is indeed important, but still does not explain everything, since for only a fifth of the length of a sentence (that would be "i thought i told you when im on the bridge that" for the first example and "you were just a little boy then it may not be as" for the second example) we still have a decent 59% of accuracy.

Let us emit and test two hypotheses.

Our first hypothesis is that *named entities* allow the model to distinguish corpora. After all, "Luke," "Yaz," "Kirk," and most of the named entities unambiguously refer to one of the corpora. To test this hypothesis, we have removed the following named entities, which occur more than a hundred times:

> Doctor, Don, Tardis, Time, Daleks, Jamie, Lord, Dalek, Brigadier, Master, Clara, Barbara, Susan, Ian, Amy, Sarah, Cybermen, Zoe, Commander, Lords, Sergeant, Rory, Tegan, Gallifrey, Romana, Rose, Smith, Donna, Ben, Part, Peri, Martha, Davros, Grant, Hang, Yaz, Adric, Nyssa
> Captain, Spock, Kirk, Enterprise, Doctor, Mc Coy, Jim, Scott, Lieutenant, Sulu, Scotty, Bones, Vulcan, Chekov, Starfleet, Federation, Uhura, Klingon
> Jedi, Master, Rey, Luke, Anakin, Artoo, Don, Chewie, Ren

(The reader may wonder why the notorious Darth Vader does not appear on the list. Surprisingly, his name occurs *only* 59 times in the whole *Star Wars* corpus. His popularity must be due to other reasons, darth reasons.) Here are the results:

	with named entities	without named entities
accuracy	0.83	0.77
precision Whovian	0.77	0.69
recall Whovian	0.81	0.70

Performance did drop, but not in an extraordinary way. We removed what would be the first discrimination criterion for a human, and we lost only 6% of accuracy.

Our second hypothesis is a more linguistic one. The Whovian corpus uses UK spelling, while the other two corpora use US spelling. Could this be of assistance in the classification process?

Indeed, we found 5,812 occurrences (538 distinct word forms) of specifically UK spellings in the Whovian corpus, the most frequent being:

> realise (402 times), centre (365 times), honour (277 times), travelling (193 times), defence (181 times), recognise (168 times), programme (117 times), marvellous (109 times), travelled (104 times), realised (101 times), travellers (89 times), favourite (87 times), favour (86 times), apologise (82 times), traveller (82 times), colour (80 times), metres (74 times), labour (69 times), behaviour (54 times), grey (52 times), …

We replaced the one hundred most frequent UK spellings with their American counterparts and fitted the model again. Here are the results:

	mixed UK and US spellings	US spellings only
accuracy	0.83	0.85
precision Whovian	0.77	0.78
recall Whovian	0.81	0.86

Our hypothesis was wrong since we got the opposite result: instead of deteriorating it, "normalizing" and spelling has, in fact, improved classification. What may have happened is that the model realized that the forms "realise" and "realize" (and all other pairs of forms) are instances of the same word, and this improved its understanding of the text. But is it ethical to transform the corpus in this way? After all, British spelling is part of the Whovian heritage; we should refrain from infringing on its integrity.

Among our two hypotheses, the first was correct, and we could quantify the impact of the loss of the discriminative feature. The second was a blatant failure, allowing us to improve our model slightly but by compromising our ethical standards.

15.2 Recurrent Neural Networks

> you don't need to understand anything about the specific architecture of an LSTM cell; as a human, it shouldn't be your job to understand it.
>
> François Chollet, *Deep Learning with Python*

When you feed a feedforward neural network with a sentence, the sentence becomes a binary vector, so the order of words is lost. A new kind of layer has been invented to re-introduce the order of words into the network's internals: the *recurrent* layer. Networks using recurrent layers are called *recurrent neural networks* (RNN). A recurrent layer is such that the layer's output calculated during propagation for an input n is used to calculate the same layer for input $(n+1)$. To be more precise, in a feedforward network we have for every dense layer d a matrix W such that for $\mathbf{x}$ received by the previous layer, layer d will take the product $W \cdot \mathbf{x}$ and a bias $\mathbf{b}$, apply the activation function σ to it, and send $\sigma(W\mathbf{x} + \mathbf{b})$ to the next layer. In a recurrent layer, we distinguish inputs chronologically by an index t: $\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}$, etc. There is a second matrix U such that the values sent to the next layer are given by the formula

$$\sigma(W\mathbf{x}_t + U\mathbf{x}_{t-1} + \mathbf{b}),$$

so that the value produced by the previous input (and therefore also the one by the input before the previous input, and so on) is involved in the calculation of the current input, this also means that depending on the previous item, the propagation of input may produce different values.

This sounds very nice, but what does a cell really learn about the past? The influence of the previous cell is combined with the values of the new cell, and the whole thing goes through the activation function before being sent to the next cell (and to the output). The influence of the preceding input is limited, and one of the earlier is practically nonexistent. This problem is called the *vanishing gradient issue* [2]. We may also have the contrary: the influence of the preceding inputs may be so overwhelming that the gradients explode.

To avoid these problems, Hochreiter & Schmidhuber [8] introduced a very sophisticated recurrent layer called LSTM (*Long-short term memory*). In Fig. 15.4, the reader can see a (simplified) illustration of the LSTM's internals. The idea is the following: grey boxes are the graphical representation of the mesh between two

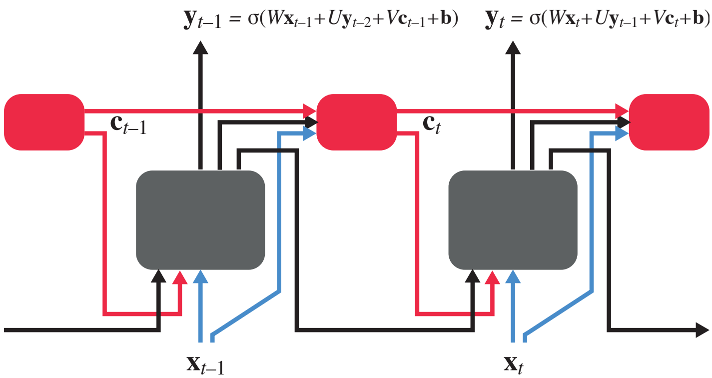


Fig. 15.4 A diagram illustrating the structure of an LSTM layer

layers: from a vector $\mathbf{x}_t$ of the lower layer, we calculate what is sent to the upper layer, namely $\mathbf{y}_t$ (beware the notation: $\mathbf{x}_t$ is *not* the t-th coordinate of vector $\mathbf{x}$, but the input that arrives at time t, so t is indexing the individuals in the training corpus). We have $\mathbf{y}_t = \sigma(W\mathbf{x}_t + \mathbf{b})$ in feedforward networks. In recurrent networks we have $\mathbf{y}_t = \sigma(W\mathbf{x}_t + U\mathbf{y}_{t-1} + \mathbf{b})$. In Fig. 15.4, we have represented the transmission of values $\mathbf{x}_*$ in blue and the transmission of values $\mathbf{y}_*$ in black.

What is new in LSTMs is that we introduce a new sequence of boxes. We have drawn them in red color. In the figure, the input of grey boxes is from beneath, and the output is from above—in the case of red boxes, the input is from the left, and the output is from the right. Red boxes are like the CIA: they are informed about everything and connected to all information channels, so they miss no information. Indeed, they receive as input $\mathbf{x}_{t-1}$, $\mathbf{y}_{t-1}$, and a special value that is transmitted from red box to red box (CIA's secret channel). We call this value $\mathbf{c}_{t-1}$ ("c" as in "carry," a nice mnemonic trick is to think of red color = blood $\to$ Stephen King's *Carrie*). The output $\mathbf{c}_t$ of a red box goes to the next red box, but also to the next grey box, so that the calculation of the layer's output value becomes

$$\mathbf{y}_t = \sigma(W\mathbf{x}_t + U\mathbf{y}_{t-1} + V\mathbf{c}_t + \mathbf{b}),$$

where V is a third matrix. The dimensions of red-box outputs are not necessarily the same as the ones of $\mathbf{x}$ or $\mathbf{y}$. It is the job of matrices W, U, and V to produce vectors of the same dimension as the bias $\mathbf{b}$ so that we can add them and apply the activation function.

The reader is probably wondering what happens inside a red/CIA box. This section's epigraph is from Chollet [3, p. 300]; it is a statement about better not sticking one's nose into what is going on inside the red box. What happens is that every red box comes with eight more matrices: $W_f, W_i, W_o, W_c, U_f, U_i, U_o, U_c$, where f stands for "forget," i and o for "input" and "output," and c for "memory cell." The task of these matrices is to propagate stuff from one red box to another and into the grey box and to forget some other stuff. The action of propagating is a solution to the vanishing gradient issue, and the action of forgetting is a solution to the exploding gradient problem. And because of this forgetting/remembering metaphor, the system is called "long-short term memory": when W_f and U_f are active, memory acts like short-term human memory[4]—when the red line is used, memory acts like long-term human memory.

This all sounds like a nice intellectual exercise. What is miraculous is that it actually works: we have 11 matrices (W, U, V and the W_*, U_* for $* \in \{f, i, o, c\}$) with random values when we start, and after some epochs, these matrices cooperate like musicians in an orchestra and accomplish the model's task.

A flaw of LSTM layers one may observe is that data are read in one direction so that an input can be influenced by the previous ones but not by the following ones. An extension of the LSTM layer has been defined to address this issue, the *bidirectional LSTM layer*. This is two LSTM layers going in opposite directions and then merging their results for every input.

15.2.1 An Example in Python

We will illustrate bidirectional LSTM layers by a classical application of these networks, namely POS tagging. We have parsed the Whovian corpus using *stanza*. We will use part of this corpus as a training corpus and then see our accuracy by comparing results obtained on a validation and a testing corpus. One may legitimately ask: What's the use of building a POS tagger out of a *stanza*-tagged corpus when we have *stanza* in the first place?

First of all, we want to show that it is possible to carry out this task using a bi-LSTM layer. Secondly, and most importantly, the example is about POS tags but any other information that is connected to the words and depends on their context can be processed in the same way.

The Whovian corpus is contained in a file of the form

```
wait    VERB
in      ADP
here    ADV
please  INTJ
...
```

where sentences are separated by a `###` line.

Here is the code of our example, inspired by Ivanov's blog[5].

```
import nltk,io,pickle,random,re

```

[4] See Cowan [4] for the difference between working and short-term memory.

[5] https://nlpforhackers.io/lstm-pos-tagger-keras

```
3  f=io.open("dw-tags.tsv",mode="r",encoding="utf-8")
4  tagged_sentences=[]
5  tmp=[]
6  for l in f:
7      if re.match(r'^([^\t\r\n]+)\t([^\t\r\n]+)',l):
8          m = re.match(r'^([^\t\r\n]+)\t([^\t\r\n]+)',l)
9          w = m.group(1)
10         t = m.group(2)
11         tmp.append((w,t))
12     elif re.match(r'^###',l):
13         tagged_sentences.append(tmp)
14         tmp=[]
15 random.shuffle(tagged_sentences)
16 tagged_sentences = tagged_sentences[:20000]
17
18 import numpy as np
19
20 sentences, sentence_tags =[], []
21 for tagged_sentence in tagged_sentences:
22     sentence, tags = zip(*tagged_sentence)
23     sentences.append(np.array(sentence))
24     sentence_tags.append(np.array(tags))
25
26 from sklearn.model_selection import train_test_split
27
28 (train_sentences, test_sentences,  train_tags,  test_tags) = \
29     train_test_split(sentences, sentence_tags, test_size=0.2)
30
31 words, tags = set([]), set([])
32
33 for s in sentences:
34     for w in s:
35         words.add(w.lower())
36
37 for ts in sentence_tags:
38     for t in ts:
39         tags.add(t)
40
41 word2index = {w: i + 2 for i, w in enumerate(list(words))}
42 word2index['-PAD-'] = 0  # The special value used for padding
43 word2index['-OOV-'] = 1  # The special value used for OOVs
44
45 tag2index = {t: i + 1 for i, t in enumerate(list(tags))}
46 tag2index['-PAD-'] = 0  # The special value used to padding
```

This code is used for data preparation. The parsed Whovian corpus contains 256,942 sentences. We shuffle them on line 15. On line 16, we filter our set and keep only 20,000 randomly chosen sentences (on a a MacBook Pro 2,3 GHz Intel Core i9 8 cores with 32 MB of RAM, model fitting of 20,000 sentences took 1 hour and 18 minutes). On lines 28–29, an *sklearn* method separates the corpus into train and test parts, together with the appropriate tags. Finally, on lines 33–46, we assign numbers to words and tags, and define number 0 for padding. This is necessary because LSTM layers require input sequences of the same size, so we take whatever size is necessary to fit all sentences, and for all smaller sentences, we pad on the right, using the `-PAD-` string. There is also an "out-of-vocabulary" symbol mapped to number 1.

```
train_sentences_X, test_sentences_X, train_tags_y, \
    test_tags_y = [], [], [], []

for s in train_sentences:
    s_int = []
    for w in s:
        try:
            s_int.append(word2index[w.lower()])
        except KeyError:
            s_int.append(word2index['-OOV-'])
    train_sentences_X.append(s_int)

for s in test_sentences:
    s_int = []
    for w in s:
        try:
            s_int.append(word2index[w.lower()])
        except KeyError:
            s_int.append(word2index['-OOV-'])
    test_sentences_X.append(s_int)

for s in train_tags:
    train_tags_y.append([tag2index[t] for t in s])

for s in test_tags:
    test_tags_y.append([tag2index[t] for t in s])

MAX_LENGTH = len(max(train_sentences_X, key=len))

from keras.preprocessing.sequence import pad_sequences

train_sentences_X = pad_sequences(train_sentences_X, \
    maxlen=MAX_LENGTH, padding='post')
test_sentences_X = pad_sequences(test_sentences_X, \
    maxlen=MAX_LENGTH, padding='post')
train_tags_y = pad_sequences(train_tags_y, \
    maxlen=MAX_LENGTH, padding='post')
test_tags_y = pad_sequences(test_tags_y, \
    maxlen=MAX_LENGTH, padding='post')

def to_categorical(sequences, categories):
    cat_sequences = []
    for s in sequences:
        cats = []
        for item in s:
            cats.append(np.zeros(categories))
            cats[-1][item] = 1.0
        cat_sequences.append(cats)
    return np.array(cat_sequences)

cat_train_tags_y = to_categorical(train_tags_y, len(tag2index))
```

Now, all sentences and their tags are represented by numbers in sequences padded by zeros. The fixed length of these sequences is `MAX_LENGTH` (in our case, 308). Line 97 converts sequences of tags to the one-hot encoded format.

```
98  from keras import backend as K
99
100 def ignore_class_accuracy(to_ignore=0):
101     def ignore_accuracy(y_true, y_pred):
102         y_true_class = K.argmax(y_true, axis=-1)
103         y_pred_class = K.argmax(y_pred, axis=-1)
104
105         ignore_mask = K.cast(K.not_equal(y_pred_class, to_ignore), \
106             'int32')
107         matches = K.cast(K.equal(y_true_class, y_pred_class), 'int32') \
108             * ignore_mask
109         accuracy = K.sum(matches) / K.maximum(K.sum(ignore_mask), 1)
110         return accuracy
111     return ignore_accuracy
```

The preparations are over; now would be the time to define the model, but one problem remains. Padding works just fine, but it gives misleading results during model fitting: when predicted tags are compared with the correct ones, most of the sequence is padding. In a sequence with 10 real tags and 261 padding tags, the comparison algorithm will conclude that at least 261 positions of the predicted sequence are correct, namely the padding ones. This is absurd, so we have to define a special metric that will ignore padding, and this is what lines 100–111 do.

```
112 from keras.models import Sequential
113 from keras.layers import Dense, LSTM, InputLayer, Bidirectional,\
114     TimeDistributed, Embedding, Activation, Masking
115 from tensorflow.keras.optimizers import Adam
116
117 model = Sequential()
118 model.add(InputLayer(input_shape=(MAX_LENGTH, )))
119 model.add(Embedding(len(word2index), 128, mask_zero=True))
120 model.add(Masking())
121 model.add(Bidirectional(LSTM(256, return_sequences=True)))
122 model.add(TimeDistributed(Dense(len(tag2index))))
123 model.add(Activation('softmax'))
124
125 model.compile(loss='categorical_crossentropy',
126               optimizer=Adam(0.001),
127               metrics=['accuracy',ignore_class_accuracy(0)])
128
129 history = model.fit(train_sentences_X, to_categorical(train_tags_y, \
130     len(tag2index)), batch_size=128, epochs=20, validation_split=0.2)
```

At last, we define the model. It uses an input layer (with `MAX_LENGTH` nodes, so that it can read even the longest sentence), an embedding layer (which will create an ad hoc embedding of 128 dimensions), a masking layer (for processing paddings), a bidirectional LSTM with 256 nodes since it will process the 128-dimensional sequences twice, once from left to right and once from right to left, a *time distributed* layer, which is a special case of the dense layer applied to every element of the sequence,

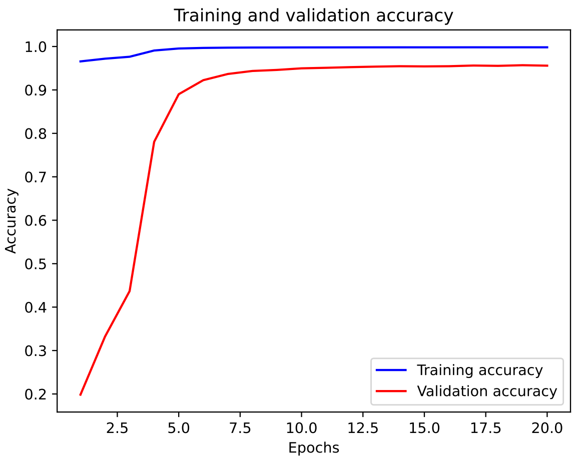


Fig. 15.5 Training and validation accuracies of the LSTM neural network example

and finally a softmax activation layer that will output the probability distribution of the tags predicted.

In the compilation (lines 125–127), we use categorical cross entropy (since we have a multi-class classification), the *Adam* optimizer, and two metrics: regular accuracy and our specially defined accuracy that ignores paddings (the `0` argument stands for the numeric value to which to map the `-PAD-` symbol).

On lines 129–130, we launch model fitting on the train corpus with a validation split of 20%, a batch size 128, and 20 epochs.

```
131 val_acc = history.history["val_ignore_accuracy"]
132 acc = history.history["val_accuracy"]
133
134 import matplotlib.pyplot as plt
135 loss = history.history["loss"]
136 val_loss = history.history["val_loss"]
137 epochs = range(1, len(loss) + 1)
138 plt.plot(epochs, acc, "r", label="Training accuracy")
139 plt.plot(epochs, val_acc, "b", label="Validation accuracy")
140 plt.title("Training and validation accuracy")
141 plt.xlabel("Epochs")
142 plt.ylabel("Accuracy")
143 plt.legend()
144 plt.show()
```

This code will display the evolution of accuracy (training and validation) during fitting. The result can be seen in Fig. 15.5. As we see, validation accuracy starts quite low, but after 10–12 epochs, it attains a plateau at around 95%.

```
145 from sklearn.metrics import classification_report
146
147 y_pred = model.predict(test_sentences_X, batch_size=64, verbose=1)
```

```
148 y_pred_bool=[]
149 for SEN in y_pred:
150     tmp=[]
151     for WOR in SEN:
152         highest_tag = np.argmax(WOR)
153         tmp.append(highest_tag)
154     y_pred_bool.append(tmp)
155 y_pred_bool=np.array(y_pred_bool)
156 tags_sorted=sorted(tag2index.keys(), key=lambda x:tag2index[x])
157 BIG_corr=[]
158 BIG_pred=[]
159 for I in range(len(test_tags_y)):
160     for J in range(len(test_tags_y[I])):
161         if test_tags_y[I][J] != 0:
162             BIG_corr.append(test_tags_y[I][J])
163             BIG_pred.append(y_pred_bool[I][J])
164
165 BIG_corr=np.array(BIG_corr)
166 BIG_pred=np.array(BIG_pred)
167
168 print(classification_report(BIG_corr,BIG_pred,\
169     target_names=tags_sorted,digits=4))
```

We will use the same `classification_report` method as in the previous example. The difference is that now our inputs are sentences; therefore, for each sentence, the output is a list of tags, while the report method expects the output to be a unique class for each sentence. What we need are statistics on POS tags [16], and not on sentences. Therefore, we merge all sentences on lines 157–163 by avoiding padding in test sentences (the `-PAD-` remains because some predictions contain it, always at the wrong slots since all correct padding slots have been removed). Here are the results

	precision	recall	f1-score	support
-PAD-	0.0000	0.0000	0.0000	0
PRON	0.9917	0.9919	0.9918	7755
DET	0.9822	0.9843	0.9833	3194
ADP	0.9486	0.9538	0.9512	2943
AUX	0.9719	0.9925	0.9821	4638
PUNCT	0.9993	0.9997	0.9995	9972
ADJ	0.8925	0.8828	0.8876	2192
SYM	0.0000	0.0000	0.0000	2
INTJ	0.9375	0.9557	0.9465	1241
NUM	0.9558	0.9257	0.9405	350
CCONJ	0.9976	0.9952	0.9964	831
PROPN	0.8495	0.6997	0.7674	1895
PART	0.9780	0.9800	0.9790	1499
VERB	0.9587	0.9324	0.9454	6003
X	0.0000	0.0000	0.0000	2
ADV	0.9409	0.9132	0.9268	3031
NOUN	0.8555	0.9312	0.8918	5379
SCONJ	0.8629	0.8556	0.8592	478
accuracy			0.9545	51405

The support is the number of POS tags that have been predicted. The results are very satisfactory. An accuracy of 95.45% is quite impressive. The only significantly lower performance is the recall of proper nouns (only 70%). As the corpus has been lowercased, detecting proper nouns that do not appear in the training corpus is difficult.

15.3 Encoder-Decoder Networks and Attention

Let us tackle the task of machine translation through neural networks. Using a feed-forward network is out of the question since word order is crucial when translating. Indeed, confusing the sentences

> Ὁ ἄνθρωπος ἔφαγε τὸν μουσακᾶ. (*The man ate the moussaka.*)
>
> Ὁ μουσακᾶς ἔφαγε τον ἄνθρωπο. (*The moussaka ate the man.*)

should be avoided at all costs (the second sentence refers to the science-fiction movie *The Attack of the Giant Moussaka*,[6] 1999, a monster movie in which Godzilla's Greek counterpart is a 20-meter-high moussaka). To preserve information on word order, recurrent neural networks are necessary. Can we use an RNN to map each word with its translation, as we did for POS tags? Certainly not, since word order is language-specific. A usual RNN, as in the previous section, will not do. To solve this problem, Kalchbrenner & Blunsom [10] had the idea of using first a recurrent network to encode information in the source language and then generate more nodes as long as necessary to translate the sentence in the source language by providing a sentence in the target language.

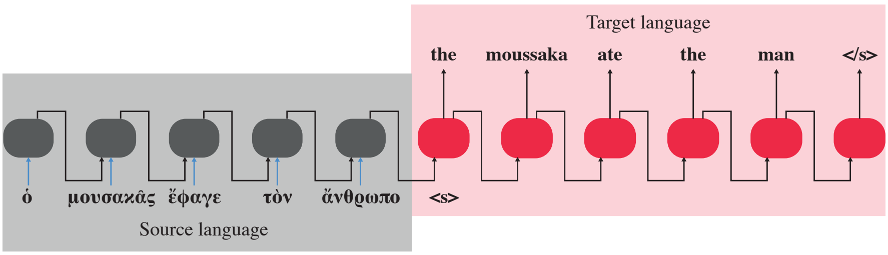


Fig. 15.6 An Encoder-Decoder network for translating “ὁ ἄνθρωπος ἔφαγε τὸν μουσακᾶ”$_{\text{EL}}$ into “the moussaka ate the man“

In Fig. 15.6, the reader can see such a network. The idea is that on the left-side (grey) part, the RNN (an LSTM or some other kind of RNN) is gathering information about

[6] The storyline on IMDb https://www.imdb.com/title/tt0287320 is a pure delight: “On the verge of the new millennium, against the backdrop of the starry Athenian sky, a twist of fate, paired with copious amounts of radiation, give birth to an unstoppable menace. As a result, what was once a scrumptious, famous Greek delicacy has now transformed into a piece of moussaka of gargantuan proportions, roaming free the streets of Athens, wreaking havoc and spreading panic.”

the source-language words in the right order. Then, when the sentence in the source language has been read, a sentence separation mark `<s>` is provided to the next cell (red in the figure), which, according to information that has been accumulated from the sentence on the left, will output the first word of the translated sentence. This word will be communicated to the next layer, but also to the next cell, whose job is to translate the next word, and so on, until the last word of the original sentence has been translated, in which case the last right-side cell will output a sentence-end mark `</s>`.

This sounds nice, and we know that with a good LSTM, we can transmit much information from one side of the cell sequence to the other. The *entire* information on the source sentence must be transmitted through the last grey cell. Imagine a sentence such as Proust's "But instead of simplicity, it was to ostentation… (followed by ☞ 91
255 more words)"—this is beyond any LSTM cell, even a huge one. The problem is that the last words of the target sentence have to rely on whatever information was gathered in the grey part and on how this information evolved in the red part. This is exacerbated when the two languages have opposite word orders, and the last words of the target language sentence depend mainly on the first words of the source language sentence, as between English (SVO) and Japanese (SOV) or Yoda (OSV). ☞ 95

What we need is some way to connect parts of the target sentence with *relevant* parts of the target sentence, wherever this may be, so that when, for example, we need to translate the verb in the target sentence, we can focus our attention on the verb in the source sentence. But how do we know where it is? How do we know where to focus our attention when processing a specific part of the target sentence?

In the previous paragraph, we used the word "attention" twice because the solution to this problem, given by Bahdanau *et al.* [1] is an improvement to the Encoder-Decoder layer, called *attention*.

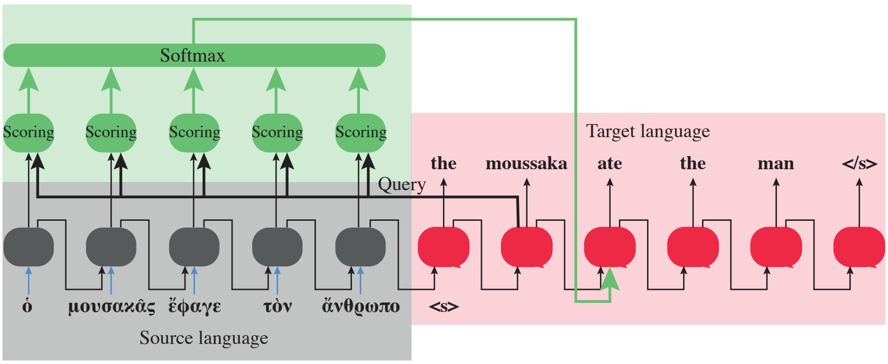


Fig. 15.7 An Encoder-Decoder network of Fig. 15.6 with an attention mechanism

In Fig. 15.7, we have equipped the network of Fig. 15.6 with an attention mechanism. To understand how it works, the reader must follow the fat line that leaves the red box of the word "moussaka." Being at that location means encoding has finished, decoding has started, and the first word, namely "the," has been output. There is already a difference: in the simple Encoder-Decoder network, the output of a grey cell (in the encoder) was only injected into the next cell. In Attention-equipped networks, this output is also sent to a mini-layer that we drew in green color. This layer is not a layer of the main network! It is a mini-feedforward network. It receives input from the encoder cells and waits. Follow the fat line that leaves the red "moussaka" cell and goes leftwards. It is called a "query," and it will transmit the red cell's output to every green cell. Green cells will examine the two inputs (one from the encoder and one from the decoder) and calculate a score. Once this is done, and as this score may take arbitrary values, we send all green cells' output to a softmax layer that turns them into a probability distribution, which is then sent to the next red cell (the one that will produce the word "ate"), together with the output from the "moussaka" cell.

The idea is that when leaving "moussaka," the query will knock at all green doors. There are as many green doors as words in the source sentence. Green doors will open, will examine the query, and will score the relevance of the query with respect to their context (remember that on the left, we may have an LSTM so that there is context information available from preceding cells or a biLSTM, in which case there is context information from both sides). Once the green cells have solemnly, sincerely, and truly evaluated the mutual relevance of the two inputs, they will transmit a score, which will be softmaxed and then sent as input to the next red cell.

15.4 Transformers and Self-Attention

> All You Need Is Love
>
> John Lennon (1967)

> Attention Is All You Need
>
> Ashish Vaswani *et al.* (2017)

The principle of attention can be applied to a single sequence of inputs so that the attention mini-network injects information from one part of the sequence to the other, for example, where there are co-references or syntactic dependencies. This kind of attention is called *self-attention*. Vaswani *et al.* in the paper [1] that we mentioned in the epigraph introduced something even more powerful: the notion of *multi-head attention*. In this case, we have not one but several attentions being calculated simultaneously. Readers may wonder how we know they are different and represent useful insights about the text. We don't. All attention heads are initialized randomly, then trained, and sometimes a miracle happens: like hunting dogs, they sniff out the game and find specific kinds of connections between words of the same sentence. Like the dog who doesn't know that the animal he has sniffed is called a partridge, the attention head detects co-references, or syntactic dependencies, without knowing the connection's nature and looking specifically for a given phenomenon. But it works all the same.

Transformers are non-recurrent networks that use self-attention. As we already mentioned, non-recurrent networks do not encode the order of words. To compensate for this, transformers use *positional encoding*, and adding numeric information to words helps recover their relative position in the sentence whenever needed.

Transformers also use the encoder-decoder principle but without the recurrent features.

15.4.1 Implementations of Transformers

More than any other NLP resource, transformers have experienced an explosion in popularity. Increasingly powerful and sophisticated models have been released in the last few years, and their expansion is not about to stop. The Web page of the *Hugging Face* community[7] contains a list of 223 (as of October 2023) transformer models, among which the popular BERT (Bidirectional Encoder Representations from Transformers) [6] (together with its French counterparts: the cheesy CamemBERT [12] and the romanesque FlauBERT [11]) and the famous GPT (Generative Pre-trained Transformer) [13] and GPT-2 [14].

To illustrate the dazzling progress of transformer models in the last four years, we will use code taken from Hagiwara [7, pp. 203–206] that uses the transformer as a language model and calculates the most probable word following a text fragment. By iterating this process, we get paragraphs of 100 words out of a given initial sentence.

Here is the code, using the *transformers* Python library and loading the *Transformer-XL* model [5] from Google Brain (1.4 GB):

```
import torch

def top_k_top_p_filtering(logits, top_k=0, top_p=0.0,\
    filter_value=-float('Inf')):
    assert logits.dim() == 1
    top_k = min(top_k, logits.size(-1))  # Safety check
    if top_k > 0:
        indices_to_remove = logits < torch.topk(logits, \
            top_k)[0][..., -1, None]
        logits[indices_to_remove] = filter_value

    if top_p > 0.0:
        sorted_logits, sorted_indices = torch.sort(logits, \
            descending=True)
        cumulative_probs = torch.cumsum(F.softmax(sorted_logits, \
            dim=-1), dim=-1)
        sorted_indices_to_remove = cumulative_probs > top_p
        sorted_indices_to_remove[..., 1:] = \
            sorted_indices_to_remove[..., :-1].clone()
        sorted_indices_to_remove[..., 0] = 0
        indices_to_remove = sorted_indices[sorted_indices_to_remove]
        logits[indices_to_remove] = filter_value
```

[7] https://huggingface.co/docs/transformers/index

```
23         return logits
24
25     def sample_token(output):
26         logits = output[..., -1, :].squeeze(0)
27         logits = top_k_top_p_filtering(logits, top_k=10)
28         log_probs = torch.softmax(logits, dim=-1)
29         token = torch.multinomial(log_probs, num_samples=1)[0]
30         return token
31
32     from transformers import AutoModelWithLMHead, AutoTokenizer
33     tokenizer = AutoTokenizer.from_pretrained('transfo-xl-wt103')
34     model = AutoModelWithLMHead.from_pretrained('transfo-xl-wt103')
35     generated = tokenizer.encode("The Doctor entered the Tardis and")
36     context = torch.tensor([generated])
37     past = None
38
39     for i in range(100):
40         output = model(context, mems=past)
41         token = sample_token(output.prediction_scores)
42         generated.append(token.item())
43         context = token.view(1, -1)
44         past = output.mems
45
46     print(tokenizer.decode(generated))
```

Lines 3–30 contain Hagiwara's code for extracting from the output of the model the next most probable word (see [7, pp. 204–205] for more information). Line 32 loads the classes that allow the use of transformers as language models. They are far from being the only application of these models, but they are what we need for this task. Lines 33–34 load the Transformer-XL model. Lines 35–36 encode our initial sentence, "The Doctor entered the Tardis and," which we already used on p. 406, so that it can be supplied as input to the transformer.

Lines 39–44 consist of a loop where we apply the model to the previously generated sentence, convert the output into a token, and add this token to the generated sentence (always encoded). Line 46 decodes the final result. In the case of Transformer-XL, this result is:

> The Doctor entered the <unk> and entered <unk>, where they would meet in the next. and meet in <unk>, where they would meet in a bar with several other characters from the show in the bar. <eos> The Doctor and Mary are both from the fictional town of Wormwood Scrubs. "I'm not going to leave Wormwood Scrubs," said Mary. "I don't want to go to a different place. They're just about to leave. <eos> Mary is the daughter of a local surgeon who has been diagnosed with Lyme disease. <eos>

We notice that the word Tardis was not recognized and that, in the beginning, the model has a hard time avoiding vicious circles, and only after the <eos> mark does it really start a narration. From then on, it seems to narrate a sitcom scenario about two doctors. This is undoubtedly a promising result.

Let us compare it with GPT-2, available through the *transformers* package (3.5 GB). Only the following lines have to be changed in the code:

```
33     tokenizer = AutoTokenizer.from_pretrained('gpt2-large')
34     model = AutoModelWithLMHead.from_pretrained('gpt2-large')
```

and

```
        output = model(context, past_key_values=past)
        token = sample_token(output.logits)
        generated.append(token.item())
        context = token.unsqueeze(0)
        past = output.past_key_values
```

The results are impressive:

> On the Doctor's command, the Tardis started up again, and the Doctor was shocked to see the Doctor was wearing a red and blue suit, which had been given to him by Amy. The Doctor asked what was wrong, but the Tardis's systems were still working. Amy explained that the Doctor had found some "fascinating data" in the console, and asked him to

GPT-2 knows the Tardis and Amy. We have an unnatural repetition of "the Doctor," where a human would use pronouns. But otherwise, there is a real story unfolding, brutally stopped by our word number limitation. We also notice that the beginning of the text is not the sequence we requested. At some point, the generated text deviated from the phrasing we requested.

To close this section, let us give ChatGPT's answer (as of May 2023) to the query "Complete the sentence "the Doctor entered the Tardis and" by exactly 100 words":

> The Doctor entered the TARDIS and felt the familiar surge of excitement course through their veins. With a knowing smile, they closed the door behind them, sealing themselves within the timeless sanctuary. The console room hummed with energy as the Doctor's hands danced across the controls, navigating the intricate pathways of time and space. The engines roared to life, propelling the blue police box into the boundless expanse of the universe. Adventures awaited, mysteries beckoned, and the Doctor, ever the intrepid traveler, eagerly embraced the unknown, ready to fight for justice and protect the wonders of existence.

Could a human author express this better? One can legitimately ask the question. Still, to play the devil's advocate, why "their veins"? And who are they who "closed the door behind them," on line 2? There is no other person mentioned, so here is a narrative flaw. But is it so? In fact, no. ChatGPT is only using gender-inclusive language and is right doing so since we didn't specify during which Doctor's regeneration the event occurred so that the Doctor could equally well be a he or a she. Asking ChatGPT, "Why did you write "they" in the previous text, referring to the Doctor?" we got the reply:

> I apologize for any confusion caused by the use of "they" to refer to the Doctor. In recent years, there has been a growing recognition and acceptance of using gender-neutral pronouns in English to accommodate individuals who do not identify strictly as male or female. In this context, "they" is used as a gender-neutral pronoun to refer to the Doctor, acknowledging their diverse and fluid identity. This inclusive language allows for a broader representation and respects the individuality of the character.

So, it was indeed gender-inclusiveness. Nevertheless, it was not specific to the ambivalence of the Doctor's gender but only general political correction. So much for ChatGPT's knowledge of the Whovian universe.

In 2023, we take ChatGPT's answers for granted and forget that there is only a gigantic transformer network behind them, pre-trained on massive data from the

Internet. What will happen next? Is this the technological singularity that will lead to humanity's extinction, as predicted by Hawking[8] in 2014? To that, ChatGPT replies:

> No, I am not the technological singularity. I am an artificial intelligence language model developed by OpenAI called ChatGPT, based on the GPT-3.5 architecture. The technological singularity refers to a hypothetical point in the future when artificial intelligence surpasses human intelligence and leads to rapid, exponential advancements in various fields. While I am an advanced AI model, I am not the singularity itself. I am here to assist and provide information to the best of my abilities within the scope of my training.

Still, the sentence "I am here to assist" is chilling…

15.5 Further Reading

15.5.1 Literature

Zillions of blogs on the Web explain this and that deep learning topic. There are also many books on the subject, but there are few really good ones that can explain, demystify, and inspire.

In the author's opinion, the best book on deep learning in general (but with quite a few chapters on NLP) is Chollet [3]. It is clear, crisp, accessible, and provides examples in Python *keras* (of which the author is the main developer).

The latest edition of Jurafsky & Martin [9] also covers quite a few topics in deep-neural NLP, without code but with quite clear and accessible mathematics.

Hagiwara [7] could have been yet another of the many "write-this-code-to-get-that-result" books, but is way beyond that. Even without a single mathematical formula (already stated on p. 3 of the book), it is a pearl in its capacity to explain complex networks.

Deep learning approaches in NLP evolve so fast that every book is out of date as soon as it is published. Publishers such as Manning are now releasing books that can be bought after the completion of the first few chapters and that are constantly updated so that, at least in theory, one is informed about the latest achievements permanently. But being informed and understanding are two different issues, and understanding sometimes bears a need for recoil, incompatible with the excitement of perpetual novelty. It's up to the reader, who has an embarrassment of riches to choose from, to ensure that this embarrassment doesn't become anxiety with deleterious effects.

15.5.2 Science Fiction

In this chapter, we mentioned the *technological singularity*, which comes to mind as soon as we converse with ChatGPT.

[8] `https://www.bbc.com/news/technology-30290540`

The theme of post-singularity humanity has been quite popular in the last twenty years. Be it Charles Stross' *Singularity Sky* (2003), *Rainbows End* by Victor Vinge (2006), or *The Transhumanist Wager* by Zoltan Istvan (2013), the perspective of machines taking over humanity, or humans progressively or brutally becoming machines, becomes, if not inevitable, at least more and more likely.

Here's a look behind the scenes. In 1986, Victor Vinge wrote *Marooned in Realtime*, a novel in which the main characters are survivors of a technological singularity (or alien invasion).

But Vinge didn't stop there. Seven years later, in 1993, he participated in the dead-serious NASA Conference *Vision-21. Interdisciplinary Science and Engineering in the Era of Cyberspace*. His presentation was entitled "The Coming Technological Singularity: How to Survive in the Post-Human Era." It is not only the title that sounds like a post-apocalyptic movie scenario—here is the abstract of this scientific publication (emphasis is ours):

> Within thirty years, we will have the technological means to create superhuman intelligence. **Shortly after, the human era will be ended.**
>
> Is such progress avoidable? If not to be avoided, can events be guided so that we may survive? These questions are investigated. Some possible answers (and some further dangers) are presented. [17, p. 11]

Among the participants at this conference was Hans Moravec, a futurist and advocate of the transhumanist movement, which calls for enhancing longevity and cognition by technology. This is not fiction anymore, and, like in the case of ChatGPT, seeing science fiction's (sometimes dark and sometimes bright) predictions become a reality leaves us with mixed feelings. Not to mention that the conference was held precisely thirty years ago, the delay mentioned in Vinge's deadly abstract.

Alarmism set aside, how can we conclude a section on the technological singularity when we know nothing about what will happen in the future? Let us hope for the best, and, as the Doctor said more than once, *Geronimo!*

References

[1] Dzmitry Bahdanau et al. "End-to-end attention-based large vocabulary speech recognition". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, pp. 4945–4949.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent Is Difficult". In: *IEEE Transactions on Neural Networks* 5 (1994), pp. 157–166.

[3] François Chollet. *Deep Learning with Python*. 2nd ed. Manning, 2021.

[4] Nelson Cowan. "What are the differences between long-term, short-term, and working memory?" In: *Prog. Brain. Res.* 169 (2008), pp. 323–338.

[5] Zihang Dai et al. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2978–2988.

[6] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of NAACL-HLT 2019*. 2019, pp. 4171–4186.

[7] Masato Hagiwara. *Real-World Natural Language Processing*. Manning, 2021.

[8] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9 (1997), pp. 1735–1780.

[9] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Third Edition Draft, https://web.stanford.edu/~jurafsky/slp3/. 2023.

[10] Nal Kalchbrenner and Phil Blunsom. “Recurrent Continuous Translation Models”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1700–1709.

[11] Hang Le et al. “FlauBERT: Unsupervised Language Model Pre-training for French”. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. 2020, pp. 2479–2490.

[12] Louis Martin et al. “CamemBERT: a Tasty French Language Model”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7203–7219.

[13] Alec Radford et al. “Improving Language Understanding by Generative Pre-Training”. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf. 2018.

[14] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf. 2019.

[15] Sandro Skansi. *Introduction to Deep Learning*. Springer, 2018.

[16] “Universal Dependencies”. https://universaldependencies.org.

[17] Victor Vinge. “The Coming Technological Singularity: How to Survive in the Post-Human Era”. In: *Vision-21. Interdisciplinary Science and Engineering in the Era of Cyberspace*. https://ntrs.nasa.gov/api/citations/19940022855/downloads/19940022855.pdf. NASA.

Chapter 16
Hints and Expected Results for Exercises

16.1 Chapter 1: Phonetics/Phonology

Hints for Exercise 1-1: English Accents

Task 1 requires data on the frequency of words. We, therefore, need frequency lists of English words (see p. 213). It also requires comparing pronunciation. The easiest way to do this is by comparing IPA strings. There is a free program providing exactly that: *espeak*[1]. It has many narrator voices, including Scottish English, UK English (North), and US English. ☞ 15

Task 2 is different: We look for words with maximal differences in pronunciation. This means that we need a phonetic distance between IPA strings to search for a maximum of this distance. One way of doing this is by using the IPA table of consonants and the IPA table of vowels. By identifying the position of each symbol in the table, we can take an Euclidean distance and use it as a phonetic distance. Let us take, for example, the word "brother," here are its three phonetic realizations:

Scottish	UK North	US
brˈʌðɜ	bɹˈʊðə	bɹˈʌðɚ

As we see, phonemes /b/ and /ð/ are common in the three realizations, but otherwise, they are pretty different. The "r" is similar in the UK and the US but different in Scottish (the famous "Scottish r"), the "o" is pronounced /ʌ/ in Scottland and the US, but /ʊ/ in the UK, and the final phoneme has three different realizations /ɜ/, /ə/ and /ɚ/.

Expected Results

The results of the first task are:

Most frequent homophones:
"the," "in," "i," "his," "he," "it," "with," "is," "be," "by"

[1] https://espeak.sourceforge.net/

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4_16

Most frequent six-phoneme homophones:
"against," "between," "myself," "english," "itself," "business," "second," "behind," "except," "opinion"

Longest homophones:
"thoughtlessness," "inquisitiveness," "disappointment," "insignificance," "self-possessed," "licentiousness," "thoughtfulness," "vindictiveness," "incompleteness," "abstemiousness"

Let's not fool ourselves. There is no profound linguistic reason why these words are rendered the same by *espeak*. It is possibly due to a lack of data. And what is a "US accent" in the first place? Accents differ tremendously between the Big Apple, Montgomery, Alabama, and Truth or Consequences, New Mexico. This solution to the exercise is an exploration of *espeak*'s limits rather than a real linguistic investigation.

As for the second task, to obtain a phonetic distance between phonemes, we transformed the IPA tables of consonants and vowels into Cartesian coordinate systems as in Fig. 16.1.

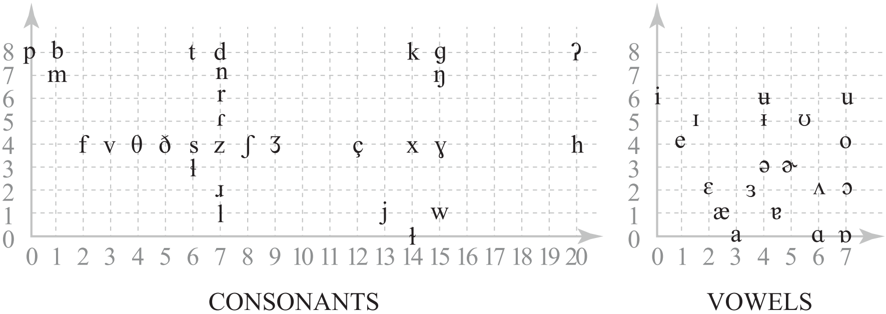


Fig. 16.1 Mapping phones to points in Euclidean space in order to define a rudimentary phonetic distance

Using that distance the following words are top-ranked with respect to their difference in pronunciation between the US and the UK:

word	US pronunciation	UK pronunciation	distance
tupelo	tuːpˈiːloʊ	tjuːpˈiːloː	72.14
tourniquet	tˈɜːnɨkɨt	tˈʊənɪkˌeː	71.57
ourselves	aɪʊɹsˈɛlvz	aʊəsˈɛlvz	69.22
algebraic	ˌældʒɪbɹˈeɪɪk	ˌaldʒɪbɹˈeːɪk	67.93
scarecrow	skˈɛɹkɹoʊ	skˈeəkɹoː	67.49
presbyopia	pɹɪsbɪˈoʊpiə	pɹɪsbɪˈoːpiə	67.11
newborn	nˈuːbɔːɹn	njˈuːbɔːn	66.67
blowfly	blˈoʊflaɪ	blˈoːflaɪ	65.92
expiate	ɛkspɪˈeɪt	ɛkspɪˈeːt	65.78
snowplow	snˈoʊplaʊ	snˈoːplæʊ	65.77

Hints for Exercise 1-2: Phonotactics of English

Syllables are an essential feature of English phonotactics, so we will explore phonotactics on the intra- and intersyllabic levels. A nice package for syllabifying English text (in US pronunciation) is *syllabify*[2].

This package uses a database of syllabified words called *CMU Pronouncing Dictionary*[3]. This induces restrictions to the vocabulary that can be analyzed. We will base our analysis on the set of words that appear both in CMU and in the frequency lists we already used in Exercise 1-1 (see p. 213), a total of 26,310 words of which we need both the frequency estimate (from the frequency lists) and the syllabic structure (given by *syllabify*).

Here is the decomposition of our Whovian named entities into phones and syllables:

lɛθ-bɹɪdʒ, zaɪ-gɑn, dɑ-lʌk, æŋ-stɹɑm, gɹæ-ʌm, jɪɹ-kɑ-nʌs, jæz, ptɪŋ, zɑ-ɑ-kɹʌs, su-zən

(the syllabification tool *melobytes*[4] proposes Le-thbrid-ge instead, we have preferred the etymological Leth-bridge, an old English family[5] originating from the county of Devon).

One can first calculate the probability of a word simply as a product of the probabilities of its syllables and see what it gives. Then, one can incorporate the conditional probabilities of the onset (or nucleus if the onset is empty) of one syllable with respect to the coda (or nucleus, if the coda is empty) of the previous syllables, as well as the probability of the initial onset (or nucleus if the onset is empty) and the final coda (or nucleus if the coda is empty).

Expected Results

By keeping track of the probabilities of syllables, initial phones, final phones as well as conditional probabilities of first syllable phone vs. last phone of the previous syllables we get

Dalek, Yrcanos, Zygon, Lethbridge, Yaz, Susan, Zaakros, Graham, Angstrom, Pting.

Since the method is analytic, one can check intermediate calculations and determine what happened. It is also possible to finetune it by giving different weights to the various types of probabilities. Finally, we shouldn't forget that we used US pronunciation because the database CMU is based on it. However, these names are either British or fictional but pronounced using the UK pronunciation in the series. Repeating the experiment with a British equivalent of CMU would be interesting.

Hints for Exercise 1-3: Tonotactics of Vietnamese

We have fetched large Vietnamese corpora from the *Wortschatz* Database in Leipzig[6], have merged them into a single file (of more than 1 GB), have cleaned it up (in particular, making it entirely NFC) and added non-alphabetic characters so that separate non-alphabetic characters surround every word. This file is available as `BIG-purified.txt.zip`. ☞ 347

---

[2] https://github.com/cainesap/syllabify

[3] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[4] https://melobytes.com/en/app/syllabication

[5] https://www.houseofnames.com/lethbridge-family-crest

[6] https://wortschatz.uni-leipzig.de/en/download/Vietnamese

Remove all numbers and replace each word with the digit representing its tone. Then, calculate the probabilities of these digits and the conditional probabilities of a digit with respect to the preceding digit. Compare them.

Expected Results

We get the following conditional probabilities which we compare with the standard probability:

Tone	$P(X \mid 1)$	$P(X \mid 2)$	$P(X \mid 3)$	$P(X \mid 4)$	$P(X \mid 5)$	$P(X \mid 6)$	$P(X)$
$X = 1$	0.38	0.39	0.37	0.41	0.37	0.38	0.39
$X = 2$	0.15	0.15	0.15	0.16	0.15	0.15	0.15
$X = 3$	0.09	0.09	0.09	0.09	0.11	0.10	0.09
$X = 4$	0.04	0.05	0.04	0.04	0.04	0.05	0.04
$X = 5$	0.19	0.20	0.21	0.20	0.20	0.20	0.19
$X = 6$	0.14	0.13	0.16	0.15	0.13	0.12	0.13

As we see in this table, the conditional probabilities never differ by more than 3% from the unconditional probabilities, so we can confidently affirm that Vietnamese tones are purely lexical and that no rule indicates that a given tone will more frequently follow some other tone.

Hints for Exercise 1-4: Classification of Voice Files

We will use the package *pyAudioAnalysis* on the files and will try, by inspection, to find one or more features that will allow us to classify the files. In French, interrogation is phonetically marked by raising the voice at the end of the sentence. “Raising” means raising the pitch and increasing intensity, i.e., energy.

Expected Results

By calculating the mean of a small number of highest energy peaks, for example, the *three* highest peaks, we get the correct results. Here are the original sentences, their translations, and the corresponding speakers:

01 “La blanquette est bonne?”$_{\text{FR}}$ ⌊Is the blanquette good?⌋[7] (Y)
02 “Je ne sais pas.”$_{\text{FR}}$ ⌊I don’t know.⌋ (D)
03 “Il est 8h50.”$_{\text{FR}}$ ⌊It is 8:50.⌋ (Y)
04 “Inutile d’attendre.”$_{\text{FR}}$ ⌊No need to wait.⌋ (Y)
05 “C’est la seule façon de faire.”$_{\text{FR}}$ ⌊It’s the only way.⌋ (D)
06 “Vous avez vu l’heure?”$_{\text{FR}}$ ⌊Have you seen the time?⌋ (D)
07 “Vous avez vu les infos?”$_{\text{FR}}$ ⌊Have you seen the news?⌋ (Y)

[7] Blanquette is a French veal dish, but here the phrase “La blanquette est bonne?”$_{\text{FR}}$ is a reference to the satirical spy film *OSS 117: Caire, nid d’espions* (2006), where it serves as Jean Dujardin’s recognition code.

08 “Je ne peux pas le faire.”$_{\text{FR}}$ ⌊I can’t do it.⌋ (Y)
09 “Il est venu hier.”$_{\text{FR}}$ ⌊He came yesterday.⌋ (D)
10 “Et demain, ce sera possible?”$_{\text{FR}}$ ⌊And tomorrow, will it be possible?⌋ (D)

16.2 Chapter 2: Graphetics/Graphemics

Hints for Exercise 2-1: Evaluating ALA-LC Transcriptions of Arabic and Greek

The ALA-LC calls these tables “Romanization tables,” which is misleading. Every transcription of some writing system into another targets a specific language, so it is not a “romanization” but an adaptation to a specific target language. Take the Latin letter “j”: it is pronounced differently in English, German, and French; therefore, every transcription must choose which language to target. In this exercise, we deal with transcriptions of American organizations. Therefore, we will consider them as targeting an English-speaking audience and evaluate them as such.

Let us denote L1 as the source language (in our case, Arabic and Greek) in its original script, L2 as the source language (in our case, English), and L1′ as the source language in the script of L2 (in our case, Arabic or Greek written in the Latin alphabet).

We will evaluate the transcriptions according to four independent dimensions:

1. *phonetic relevance*: are the L1′ graphemes appropriate? We have two subcases:
    a. the case of L1 phonemes that exist in L2. In this case, do the L1′ graphemes represent the L1 phonemes appropriately?
    b. the case of L1 phonemes that do not exist in L2. In this case, the L2 graphemes used should have phonetic realizations that are close to L1 phonemes and should not be very frequent in L2;
2. *ambiguity*: are L1′ graphemes chosen so that no ambiguity can arise? If not, how frequent are words with ambiguous transcriptions in L1?
3. *automaticity*: is the transcription method automatic, i.e., can it be done mechanically, or does it require a thorough knowledge of L1?
4. *input simplicity*: how easy is it to input L1′ graphemes on a L2 keyboard?

We will assign a mark between A and C to each one of these dimensions for each transcription table.

Expected Results

Here is a table of the ambiguities of the Arabic ALA-LC transcription, with frequencies:

Transcription	Case	Frequency	Case	Frequency	Ratio (in %)
t/h	ة	22,308,497	final ت or ه	11,709,590	34.42%
th	ث	3,274,245	ته	1,596,582	32.78%
kh	خ	5,559,596	كه	80,717	1.43%
dh	ذ	6,525,109	ده	416,177	6.00%
sh	ش	6,384,929	سه	273,202	4.10%

and here is the corresponding table for the Greek ALA-LC transcription:

Transcription	Case	Example	Frequency	Case	Frequency	Ratio (in %)
ng	νγ	σίνγκλ (single)	20,885	γγ	9,011	69.86%
nk	νκ	Φράνκ (Frank)	8,728	γκ	94,978	8.42%
nx	νξ	Μπρόνξ (Bronx)	160	γξ	139	53.51%
nch	νχ	Μανχάτταν (Manhattan)	2,537	γχ	2,857	47.03%
ps	πσ	Γκίμπσον (Gibson)	1,533	ψ	15,689	8.90%

To conclude, we mark the Arabic ALA-LC transcription with scores B/B/B/A$^-$ and the Greek ALA-LC transcription by A$^-$/A$^-$/A/A$^-$, which is far better than the Arabic ALA-LC transcription. Still, Arabic is one of the most complex languages to transcribe.

Hints for Exercise 2-2: Graphotactics of English

We will adapt our solution to Exercise 1-2 to graphemes. The main difference is the issue of syllables. If we use a syllabifier such as *spacy_syllables*[8], we get a decomposition that is phonologically oriented and very hard to use for our calculations since the "syllables" provided by this program are very irregular. Here are some examples:

un-pre-dictabil-i-ty, hy-po-bromite, serendip-i-ty, squeegee, etc.

One can hardly perform statistics on "syllables" such as "dictabil," "bromite," "serendip," etc. And they do not respect any onset-nucleus-coda scheme. This is not an isolated phenomenon: the *pyphen* and *hyphenate* packages give the same results since all these packages are based on Frank Liang's algorithm, described in his seminal Ph.D. thesis *Word Hy-phen-a-tion by Com-put-er* (Stanford 1983) [20], the algorithm used by TEX for hyphenating words in any language. The results of TEX's hyphenation algorithm have been corrected and fine-tuned for the last forty years. We cannot use them because they are based on *hyphenation* rules and not on *syllabification* rules. You are not supposed to hyphenate "serendipity" otherwise than "serendip-i-ty," this doesn't mean there are no syllables in the first part. It only means that we are not allowed to separate them when hyphenating.

Instead of giving a new definition of graphemic syllables, which would be lacunary and contentious, we will use a much more neutral approach: trigrams. And since word begin is important (and one does not start a word with just any trigram), we will also consider initial trigrams.

Expected Results

The result we get is indeed better than in Exercise 1-2:

Graham, Angstrom, Susan, Lethbridge, Dalek, Zygon, Pting, Yrcanos, Zaakros, Yaz

[8] https://spacy.io/universe/project/spacy_syllables

Hints for Exercise 2-3: Greek Car License Plate and Signs

The particularity of letters “A,” “B,” “E,” “H,” “I,” “K,” “M,” “N,” “O,” “P,” “T,” “X,” “Y,” and “Z” is the fact that they can interpreted both as letters of the Greek alphabet and letters of the Latin alphabet (not necessarily with the same phonetic correspondence, as in the case of “P” which is /p/ in English and /r/ in Greek). This restriction is due to the *Convention of Road Traffic*, which states that

> The registration number referred to in Articles 35 and 36 of this Convention shall be composed either of numerals or of numerals and letters. The numerals shall be Arabic, and the letters shall be in capital Latin characters. [7, p. 47]

To answer the question about the status of the license plate symbols, think of the definition of grapheme: a grapheme is an elementary unit of a writing system, obtained by the method of minimal pairs and used to build morphemes (minimal units of meaning) through second articulation. Is this the case on the license plate?

As for the four images, here is some additional information:

1. the Greek transcription of the English word “PARKING” is “ΠΑΡΚΙΝΓΚ”$_{\text{EL}}$;
2. “ΛΟΥΚΟΥΜΑΔΕΣ”$_{\text{EL}}$ is the Greek version of *lokma*, dough balls made with flour, dry yeast, water, and starch, and served warm with honey and cinnamon. The English transcription of this word could be “LUKUMADES.” The image displays the logo of a dessert shop[9] in Athens airport;
3. “μιγάδα”$_{\text{EL}}$ ⌊mixed-race⌋ is the name of a feminist journal[10] published in Athens ever since 2011;
4. “μετάφραση”$_{\text{EL}}$ is the Greek word for ⌊translation⌋, this the logo of a Translators’ Training Center[11] in Athens.

Expected Results

There is no second articulation in car license plates. When we say that a car has the (hypothetical) plate identifier “EPN-3498,” the letters “E,” “P,” and “N” do not form a morpheme, neither in the Greek nor the English language. Even in the rare cases where a morpheme emerges, such as “NEO,” meaning ⌊new⌋ in Greek or “ATE” in English, as in the verb “I ate,” these morphemes are purely accidental and have no connection to the car license plate.[12] Therefore, the symbols used on the plate are not graphemes. Nevertheless the four examples we give are quite different and can be discussed.

Hints for Exercise 2-4: Predictability of New Sinograms

A simple way of analyzing the structure of sinograms is by the use of the *Ideographic Description Characters* (IDC) U+2FF0 to U+2FFB. These characters describe the possible arrangements of components. Ten are binary (two components), and two are ternary (three com-

---

[9] https://www.lukumades.com/

[10] https://migada71.wordpress.com/

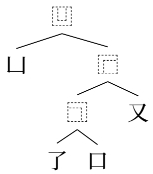
[11] https://www.metafrasi.edu.gr/

[12] In the US and some other countries it is possible to have personalized car license plates, called “vanity plates,” portraying a recognizable word or abbreviation. This is not the case in Greece.

ponents). When we combine them with Unicode characters, we get *Ideographic Description Sequences* (IDS). Here are examples of their use, as well as their relative frequencies in the BMP:

IDC	⿰	⿱	⿲	⿳	⿴	⿵
Frequency	69.22%	21.09%	0.18%	0.54%	0.76%	1.05%
Example IDS	⿰舌乚	⿱一内	⿲手分手	⿳士冗心	⿴工从	⿵几丶
Target	乱	丙	掰	悫	巫	凡
IDC	⿶	⿷	⿸	⿹	⿺	⿻
Frequency	0.09%	0.24%	3.59%	0.64%	2.49%	0.11%
Example IDS	⿶凵㐅	⿷匚久	⿸厂人	⿹勹九	⿺乙田	⿻山豩
Target	凶	匛	仄	勼	乪	豳

Note that IDCs are not combining characters in the sense that there is no requirement that a sequence consisting of an IDC and some sinograms should automatically be rendered by the target sinogram. At least the author knows of no font with this feature. Their purpose is to describe the structure of a sinogram using Unicode characters whenever possible. IDS are tree structures, as illustrated by the sequence ⿶凵⿸⿹了口又, that stands for



and describes **函**. We will use files IDS-UCS-Basic.txt and IDS-UCS-Ext-B.txt adapted from the CHISE Web site[13] [22]. Each line of these files consists of a Unicode codepoint, the corresponding character, and its IDS. The author of the CHISE files also used XML entities for
☞ 351 components not available in Unicode. We have replaced them with PUA characters, described in IDS-UCS-dict.txt.

To answer the question, we will define *patterns*, store them, and generate random patterns based on their probability distribution in the BMP. A pattern will be an IDS where we have replaced sinograms by "A," "B," "C," etc., for example ⿶凵⿸⿹了口又 belongs to the pattern ⿶A⿸⿹BCD. A pattern with n letters becomes a n-ary function from the set of sinograms to the set of IDSs—the letters stand for the arguments of this function.

For each pattern, we will calculate probabilities of all possible values of its arguments. The probability of a given pattern realization will be the product of the probabilities of its arguments. We remove from the set of combinations those that stand for characters already present in the BMP.

Once a pattern has been chosen to create a new sinogram, we take the most probable combination of values for its arguments. We repeat this process a thousand times and check how many IDSs are contained in the SIP.

Expected Results

We predict 389 SIP characters and therefore our predictability is 38.9%.

[13] https://www.chise.org/ids/

Hints for Exercise 2-5: Exotype Classification

Fig. 16.2 What can happen if you don't pay attention to the exotype you are using

First, an accurate definition of the exotype property must be given, accurate enough to identify borderline cases and cases that we will not consider as exotype (e.g., the case of Fig. 3.13(l) can very well end up not having the exotype status). Once the notion of exotype is well-defined, consider all possible ways to compare exotype typefaces, including their characteristics vs. scripts S1 and S2, their function, and their possible uses and misuses. À propos misuses, in Fig. 16.2 we show an Indian restaurant frontage. The photo on the left was taken in 2013. As the reader can see, the presumably Indian exotype is, in fact… Hebrew. On the right, the same restaurant two years later, the font has been replaced by a Devanagari exotype.

Expected Results

An adequate definition of a text written in an exotype typeface could be: "a text that is spontaneously and superficially recognized as using τ-graphs from script S2, which, modulo some time and effort needed for adaptation, can also be interpreted as τ-graphs from script S1, and deliver a message in a language using script S1".

This definition provides us with a first set of classification criteria:

1. the identity of script S2;
2. the recognition potential of script S2;
3. the time and effort needed to discover the underlying script S1;
4. the legibility of the linguistic message, written in script S1.

These are global criteria. Now we can be more specific and look at the mechanism of the typeface. Is it using authentic τ-graphs from script S2, or are they intermediate forms (between τ-graphs from script S1 and script S2)? The difference can be seen in the figure: example 3.13(d) uses authentic Arabic τ-graphs (some of them mirrored or rotated) while 3.13(e) uses specially drawn shapes that only partly correspond to Arabic shapes. Here are some more criteria:

5. what is ratio of authentic S2 τ-graphs in the text?
6. have they been rotated or mirrored?

Finally, one can classify exotypes by their potential uses.

Classification of exotypes is an ongoing project by the author. The reader can consult the Web page of the project[14] and is kindly requested to contribute by reporting more exotype fonts.

[14] https://www.fluxus-editions.fr/exotypes.php

16.3 Chapter 3: Morphemes, Words, Terms

Hints for Exercise 3-1: English and French Verb Conjugation Compared

One way of measuring complexity is by the number of states needed when writing down the FSA of the verb's conjugation. As for the forms that are the most distant, one can use, for example, the Levenshtein distance to compare the infinitive form with all other forms. This distance counts the minimum number of operations (adding a grapheme, deleting a grapheme, substituting a grapheme with another grapheme) needed to transform one string into another.

Expected Results

The verb “to teach” is the absolute winner of the FSA complexity in English: it requires 13 states and is the *only* verb in our list of 3,276 verbs requiring so many states (see Fig. 16.3). The verb form that is the farthest from its infinitive is “sought” (from the verb “to seek”). It has a Levenshtein distance of 5 since three letters have to be substituted and two added.

In French, as expected, figures are much higher: the maximum number of states is 36, for the verb “vouloir”$_{\text{FR}}$ ⌊to want⌋ (but also “souloir”$_{\text{FR}}$ ⌊to have the habit of⌋ and “douloir”$_{\text{FR}}$ ⌊to ake⌋, both obsolete) (see Fig. 16.3). As for the most distant form, it is “vécussions”$_{\text{FR}}$, the 1st person of the plural of the imperfect subjunctive of verb “vivre”$_{\text{FR}}$ ⌊to live⌋.

It is interesting to note that “to teach” and “vouloir”$_{\text{FR}}$ have a lot in common: (a) they are very common words, (b) their stem (in the sense of common prefix to all forms) consists of a single letter, (c) they both have two main paths leaving the stem: “teach” vs. “taught,” “veu” versus “vou.”

Hints for Exercise 3-2: Jules Verne and French Verbs

The earliest book by Jules Verne available on Project Gutenberg is “Cinq semaines en ballon”$_{\text{FR}}$ ⌊Five Weeks in a Balloon⌋ (1861), and his latest work on the same site is “Le village aérien”$_{\text{FR}}$ ⌊The Village in the Treetops⌋ (1901). There are forty years between the two, we have chosen as intermediate books, “L'île mystérieuse”$_{\text{FR}}$ ⌊The Mysterious Island⌋ (1874) and “Nord contre Sud”$_{\text{FR}}$ ⌊North Against South⌋ (that deals with the American Civil War).

Concerning the comparison between *Talismane*, *stanza*, *treeTagger*, and *spaCy*, the appropriate way to do it would be to take, e.g., 100 random sentences, parse them with the three packages and compare the results. As this would require significant time and effort (and some knowledge of French), we have chosen one sentence that happens to be parsed correctly by one of the four packages and incorrectly by the other three. The sentence is

> L'ingénieur entraîna son compagnon, dont la confiance en Cyrus Smith était telle qu'il ne doutait pas que l'entreprise ne réussît.

The sentence can be found in the *Mysterious Island*, line 2657 of the file `2-ile-mysterieuse.txt`. The correct results are:

- “entraîna”$_{\text{FR}}$: indicative simple past,
- “était”$_{\text{FR}}$: indicative imperfect,
- “doutait”$_{\text{FR}}$: indicative imperfect,
- “réussît”$_{\text{FR}}$: subjunctive imperfect.

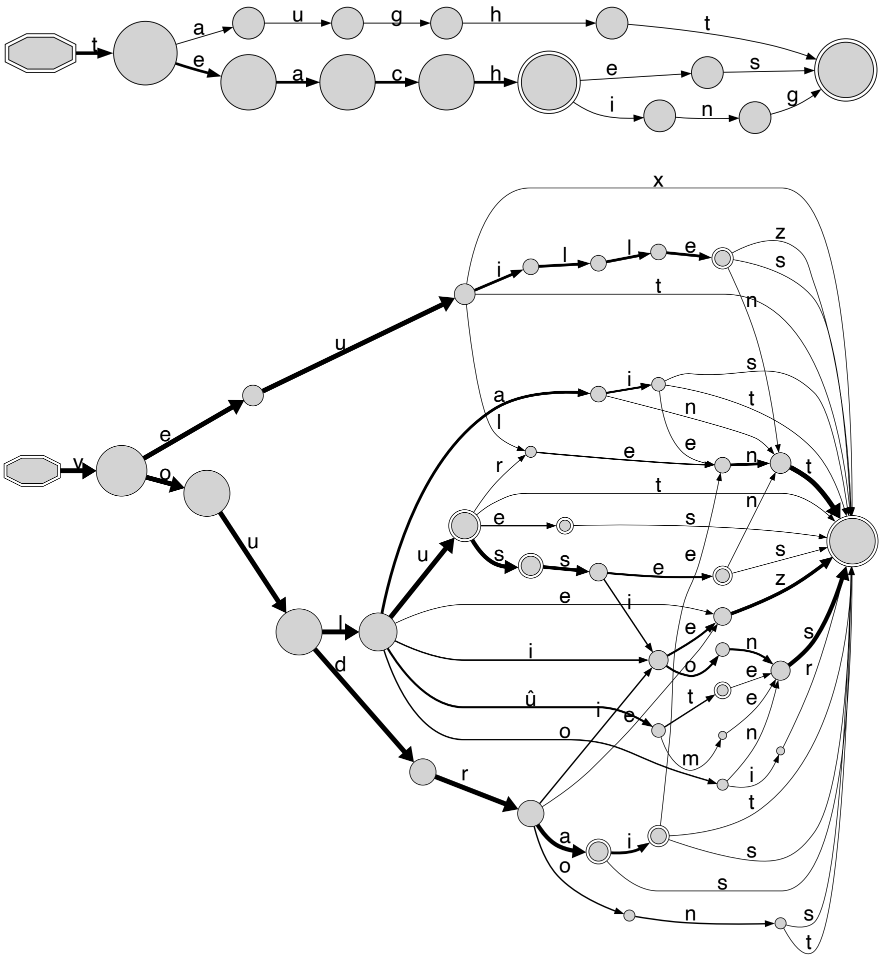


Fig. 16.3 Finite-State Automata for the most complex conjugations in English (verb "to teach") and in French (verb "vouloir")

As for the simplicity of verbs, we provide a list of 5,000 verbs `verb-frequency-list.txt` ordered by decreasing frequency. We will calculate each book's average verb rank (according to this ranking).

Expected Results

Here are the results (in percentages over all verbs):

Book	Cond.	Future	Imp.	Past p.	Present	S. past	Subj. past	Subj. pres.	Avg rank
Five Weeks...	2.68	4.42	20.15	18.95	30.4	21.84	0.68	0.88	251.97
Mysterious...	3.54	2.78	30.29	24.15	16.53	19.22	2.83	0.67	215.32
North against...	5.13	2.78	29.23	26.53	18.65	13.43	3.43	0.83	192.99
Village in the...	4.93	1.55	26.36	24.01	21.8	17.08	3.3	0.97	240.64

We can conclude that Jules Verne simplified his writing over the years, except for the last book, which is again a bit more formal. Here are our arguments:

1. a marker of formality and sophistication in French is the use of the simple past (as in "j'aimai"$_{\mathrm{FR}}$), which is a less formal register is replaced by the present perfect ("passé composé"$_{\mathrm{FR}}$, as in "j'ai aimé"$_{\mathrm{FR}}$). We see that Verne's use of the simple past has decreased from 21.84% to 19.22%, to 13.43% (almost half as 26 years earlier) and then increased again (17.08%) for the last book;
2. does the fact of having fewer occurrences of simple past mean that present perfects have progressively replaced them? This is hard to prove directly because the present perfect is actually an auxiliary ("avoir"$_{\mathrm{FR}}$ ⌊to have⌋ or "être"$_{\mathrm{FR}}$ ⌊to be⌋) followed by a past participle, but between the two, other words can occur (adverbs, negation, etc.). Identifying the perfect one would need dependency parsing, and *TreeTagger* does not provide it. But, statistically, a large part of past participles is involved in present perfects, so if we observe a decrease in past participles, this may also mean more present perfects. And indeed, past participles increase from 18.95% to 24.15%, and then to 26.53%, and only decrease slightly for the last book (24.01%);
3. finally, the same phenomenon is observed in the case of the average frequency rank of verbs: from 251.97, it goes down to 215.32 and then to 192.99, and only increases for the last book (240.64).

We have a cluster of clues implying that Jules Verne simplified his writing style between 1861, 1874, and 1887 and then returned to a slightly more sophisticated style in 1901 (four years before his death). It would be worth taking all of Jules Verne's books on the Gutenberg Project to see whether this tendency is more general.

The Jules Verne translator and specialist Volker Dehs, whom we contacted on this subject, asserted that Verne's writing became progressively simpler, sometimes even in the same book, between the first chapter and the remainder. He also mentioned a textometric study [24] that proved that words used by Verne became progressively shorter. These two arguments are compatible with our findings.

Hints for Exercise 3-3: The Combinatorics of Neoclassical Morphemes

We will find neoclassical prefixes and suffixes by calculating prefixes and suffixes of length $n \geq 3$ between any pair of words in 🗋`words.txt`. We will keep only those prefixes/suffixes that occur at least N times for different N between 1 and 20. Once a prefix has been identified, we will remove it from the word and add the remainder (if it is of length greater than 3) to a supplementary list of words. The process will be repeated until there are no word parts left. How many prefixes and suffixes of our inventory will we be able to detect, and how many false positives will we get?

Expected Results

The result can be seen in Fig. 16.4. As we see, recall decreases, and precision increases when

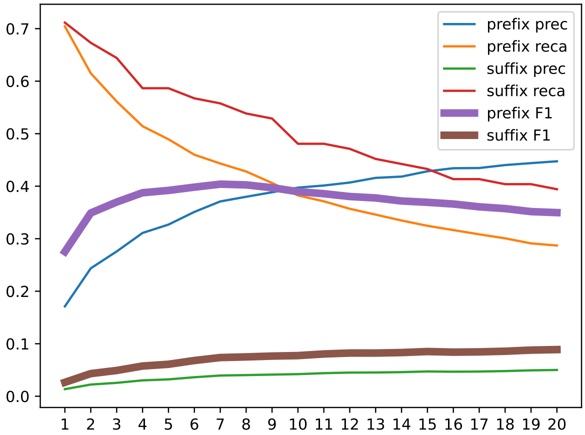


Fig. 16.4 Plot of precision, recall and F1-score for neoclassical prefix and suffix identification

frequency increases. As for the F1-score, for prefixes, it reaches a maximum of 0.40 for frequencies 7 and 8 and then drops, while the F1-score for suffixes is globally very, very low.

Given these results, we can only conclude that identifying neoclassical prefixes and suffixes by purely algorithmic means (without any additional resources on Greek and Latin morphemes) is doomed to failure.

Hints for Exercise 3-4: The Morphology of Lojban

Lojban is defined in such a way that the graphemic properties of each word determine its part of speech. Read Chapter 2 as well as sections 4.2 and 4.3 of [8][15] to find out what are cmavo, brivla, and cmene, and how they are recognized.

Expected Results

- “ni’o”$_{\text{LOJ}}$, “lo”$_{\text{LOJ}}$, “fa”$_{\text{LOJ}}$, “du’u”$_{\text{LOJ}}$, “na”$_{\text{LOJ}}$, “ka’e”$_{\text{LOJ}}$, “se”$_{\text{LOJ}}$, “se”$_{\text{LOJ}}$, “cau”$_{\text{LOJ}}$, “nu”$_{\text{LOJ}}$, “da”$_{\text{LOJ}}$, “poi”$_{\text{LOJ}}$, are cmavo;
- “seldau”$_{\text{LOJ}}$, “selmi’ecatra”$_{\text{LOJ}}$, “katna”$_{\text{LOJ}}$, “vimcu”$_{\text{LOJ}}$, “stedu”$_{\text{LOJ}}$, “xadni”$_{\text{LOJ}}$, are brivla;
- there are no cmene in this sentence (except for the translator’s name “.xorxes.”$_{\text{LOJ}}$).

[15] https://lojban.org/publications/cll/cll_v1.1_xhtml-section-chunks/index.html

Hints for Exercise 3-5: Long and Round Ess in German

As long ess and round ess depend on the word's morphology, we need a morphological analyzer that decomposes components. An ess at the end of a component has to be a round ess. Unfortunately, neither *stanza* nor *spaCy* deal with components. As of July 2023, the only Python package to manage this task is *german_compound_splitter*.

We will consider the problem of round and long as a classification problem (two classes). Use the Python package just mentioned and calculate precision and recall for each class.

As the results are not highly encouraging, we have created our own corpus out of freely available resources from the University of Tübingen, the Institut für Deutsche Sprache in Mannheim, and data provided by the Python package *german-nouns*.

Expected Results

To run the package *german_compound_splitter* you need a dictionary of German words. We provide it as file 🗋`exo03-6-corpus.dic`. You also need the standard version and the golden version of *Dichtung und Wahrheit*: 🗋`dichtung.txt` and 🗋`dichtung-gold.txt`.

The results, if we omit words starting or ending with ess (in which case the form of the ess is deterministic) are the following:

	Precision	Recall
Round ess class	0.86	0.34
Long ess class	0.86	0.99

Recall is very bad. To get better results, we decided to build our own corpus. We took a list of analyzed compound words from the University of Tübingen[16] and processed it so that we got a list of compound words 🗋`from-tuebingen.txt`. Afterwards, we took the 1,000 most frequent nouns from the Institut für Deutsche Sprache at Mannheim[17] and declined them into nominative and genitive using the Python package *german-nouns*. We added a few prepositional prefixes to the list and obtained 🗋`from-german-nouns.txt`. The results are not perfect, but significantly better than the previous ones:

	Precision	Recall
Round ess class	0.91	0.73
Long ess class	0.97	0.992

[16] https://uni-tuebingen.de/en/faculties/faculty-of-humanities/departments/modern-languages/department-of-linguistics/chairs/general-and-computational-linguistics/ressources/lexica/germanet/description/compounds/

[17] http://www.ids-mannheim.de/fileadmin/kl/derewo/DeReKo-2014-II-MainArchive-STT.100000.freq.7z

16.4 Chapter 4: Syntax

Hints for Exercise 4-1: Constituency parser comparison

The corpus we will use comes from a parser developed at CMU by Daniel Sleator[18], the sentences are contained in `samples.txt`. The first 513 sentences are grammatical, and the remaining 332 (marked by an asterisk) are agrammatical. The parsers we will use are *stanza* and *benepar*, which are part of the *spacy* package. For the latter, you need to install *spacy*, then `en_core_web_md` by writing

```
python -m spacy download en_core_web_md
```

and finally `benepar_en3` by running the following file:

```
import nltk
benepar.download('benepar_en3')
```

The exercise is about structural differences, meaning we don't care about trees having the same structure but different labels. How do we compare the structure of two trees independently of the node labels? Considering that our trees will be represented in parenthetical notation such as

```
(S (NP (DT The) (NN fact) (SBAR (IN that) (S (NP (PRP he))
(VP (VBD smiled) (PP (IN at) (NP (PRP me)))))) (VP (VBZ gives)
(NP (PRP me)) (NP (NN hope))) (. .))
```

a very naive way of comparing structures is by looking at groups of closing parentheses and comparing them. In the example above, we have first a single closed parenthesis (after the word "The"), then another one ("fact"), then another one ("that"), then a group of tw0 ("he"), etc. A "signature" of this tree would therefore be 1-1-1-2-1-1-7-1-2-3-2. We want to compare this signature with another one, which can be longer or shorter and contain the same integers. The easiest way to compare them is by the Levenshtein distance, assuming that inserting, removing, or replacing an element of a signature increases the distance by 1.

We calculate the Levenshtein distance between the trees of sentences parsed by *stanza* and *benepar* and keep the pairs of the highest distance. To distinguish between ex-equo cases, we will calculate tag frequencies (for tags only) and obtain the average tag frequency for each sentence. We will describe the three structurally most different trees between *stanza* and *benepar*.

Expected Results

We will process the five most distant trees. Here is the first one:

1. The first valid result is sentence #86, with a Levenshtein distance of 6 between the two trees:

#86. My dog, cat, horse, and mouse, and his cow left.

It is a tricky sentence explaining the difference between the two parses.

The reader can see the two trees in Fig. 16.5. The trees are indeed very different. First, *stanza* uses a tag for noun lists and includes "dog, cat, horse, and mouse" in this category. But

---

[18] https://www.link.cs.cmu.edu/link/batch.html

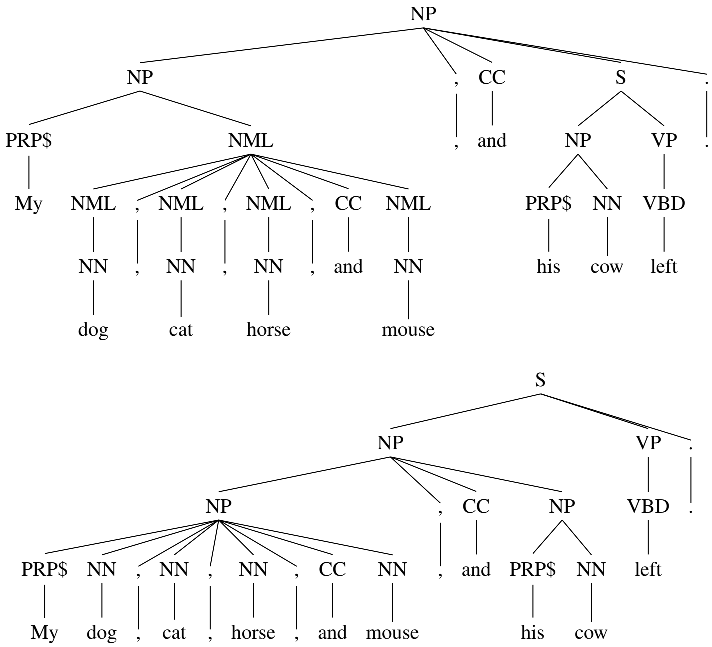


Fig. 16.5 Trees of the sentence "My dog, cat, horse, and mouse, and his cow left." as obtained by *stanza* (top) and *benepar* (bottom)

the most important difference is that *stanza* fails to realize that "left" applies to all animals mentioned and not only to the cow. On the one hand, for *stanza*, this sentence is not a sentence but just a noun phrase in two parts: the first part is "My dog, cat, horse and mouse" (which has no verb), and the second part is the sentence (!) "his cow left," which is rather strange. On the other hand, *benepar* parses the sentence correctly and identifies the list of animals as being a single NP, the subject of the VB "left."

Hints for Exercise 4-2: How well do *stanza* and *spacy* parse Yoda?

We will parse both `yoda.txt` and `adoy.txt` with *spacy* and *stanza* and we will check two things:

1. whether the root of the Yoda sentence is the same as the root of the de-yodified sentence;
2. how many dependencies the two sentences have in common.

For the latter, we will use the naive formula $\frac{\text{\#intersection of dependencies}}{\text{\#union of dependencies}}$.

Expected Results

spacy is only slightly better than *stanza* (a difference of 1.74%). As for keeping the same root, there is a significant difference between the two: *spacy* has an accuracy of 0.55 while *stanza* reaches an accuracy of 0.725. We also notice that *stanza* tends to have more extreme values: the maximum of 1.0 is attained twice (#7, #33) and the minimum of 0.1 also twice (#37, #38), while the maximum of *spacy* is 0.9 (#31) and its minimum, 0.125 (#30). In Fig. 16.6, the reader can see the distribution of the compared performances of *spacy* and *stanza*, marks denote the combinations of root identification, and numbers the sentences #s.

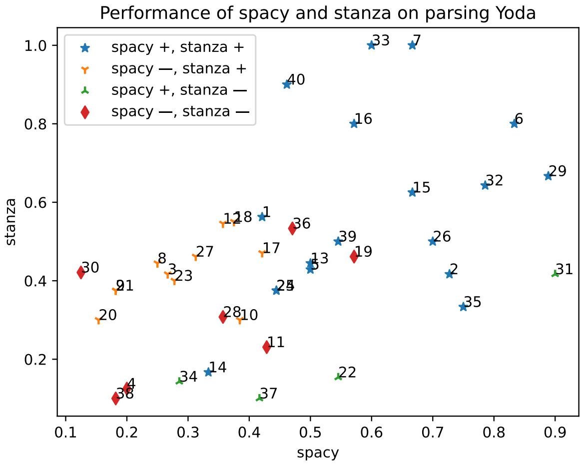
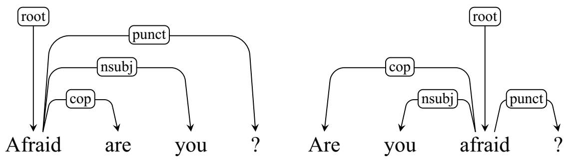


Fig. 16.6 Compared performances of *spacy* and *stanza* on the Yoda corpus

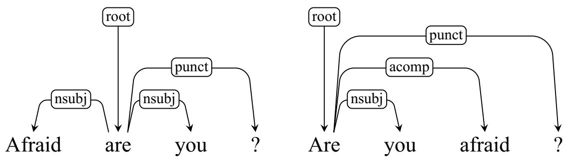
We will examine four extreme cases (#7, #37, #31, and #30). Here is the first one:

#7: Afraid are you?

This is one of the two sentences where *stanza* achieves a total success:



Here is how *spacy* handles this sentence and its de-yodified version:



For some reason, *spacy* considers "Afraid" as a subject of "are" and also "you," so the Yoda sentence has two subjects. As for *stanza*, it takes "afraid" as the root of the sentence because "are" is a copula.

Hints for Exercise 4-3: The Syntax of Lojban

We will use the *ilmentufa*[19] Lojban parser, as follows:

```
node run_camxes -m N "ni'o seldau lo selmi'ecatra fa lo du'u na ka'e
katna vimcu lo stedu se cau lo nu da poi xadni zo'u ka'e katna vimcu
lo stedu da"
```

As explained in the parser documentation, the `-m N` option produces a relatively compact syntax tree sufficient to get a global idea but not enter into details. The option `-m R` gives *all* nodes of the syntax tree based on the syntax rules described in the CLL (aka *The Complete Lojban Language* [8, §21.2]. By comparison, the former tree has 39 nodes and the latter 339 (!) nodes. The reader will need both to understand the syntax better, using the latter to disambiguate the former.

Expected Results

The official formal grammar of Lojban is written in YACC format and can be downloaded[20] from the Lojban site. The YACC grammar has rules labeled from 1 to 899 (or to 1099, depending on how the grammar is used). The most important Ninety-one rules are presented in the CLL [8, pp. 490–492]. When using *ilmentufa* with option `-m R`, one gets *all* YACC rules used, some of which carry no grammatical knowledge and exist only for practical reasons.

As a compromise between the simplistic `-m M` representation and the overwhelming `-m R` representation, we have drawn on Fig. 16.7 a tree that omits all unary rules. For example, between the node labeled “paragraphs” and its child labeled “tail-terms,” ten unary rules have been applied. These rules all have optional parts that can potentially make them non-unary. However, the optional parts are not used in this specific example. The number by which we label each edge is the *last rule* applied to obtain the result.

As in every constituent syntax tree, there are three kinds of nodes: syntax nodes (written in lowercase regular type), POS tags (uppercased), and word forms (boldened). The latter are the leaves of the tree.

Fig. 16.7 should be carefully studied and its various branches explained.

Hints for Exercise 4-4: Find Perfectly Ambiguous Sentences in English

One of the reasons why these French sentences work perfectly is the fact that in French, “le$_{\text{FR}}$” can be a personal pronoun (version 1) or a determiner (version 2). This is not the case in English. Therefore, we will try something simpler, namely a configuration

Version 1	DET	ADJ_1	NOUN_1	VERB_1
Version 2	DET	NOUN_2	VERB_2	NOUN_3

On the one hand, in version 2, NOUN_3 has to be the direct object of VERB_2, and this implies two things: (1) that VERB_2 takes a direct object, and (2) that NOUN_3 is in the plural so that it takes no determiner (a determiner between VERB_2 and NOUN_3 would make version 1 impossible).

[19] https://github.com/lojban/ilmentufa

[20] https://www.lojban.org/publications/formal-grammars/grammar.300.txt

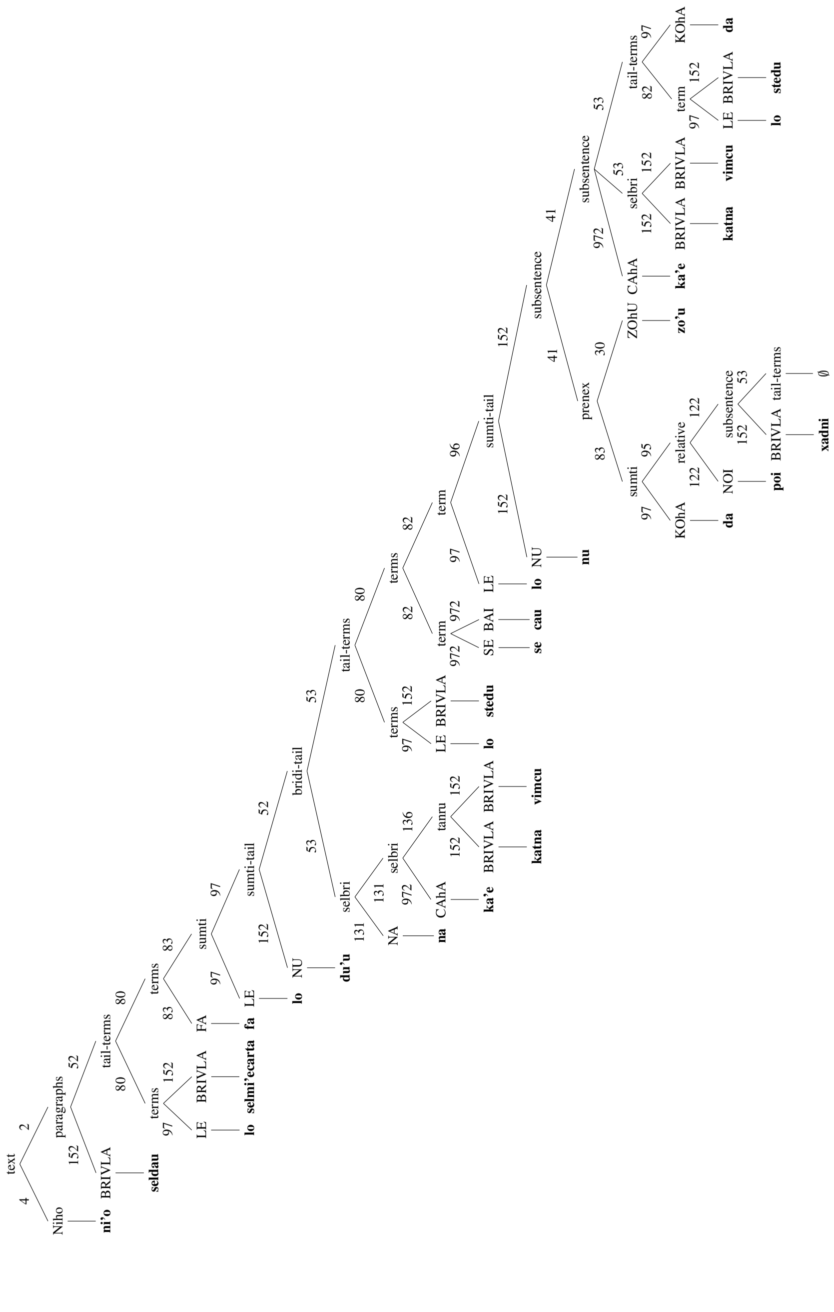


Fig. 16.7 Syntax tree of the sentence "The executioner's argument was that you couldn't cut off a head unless there were a body to cut it off from" from the *Adventures of Alice in Wonderland*. Numbers are production rule numbers from the YACC file. When multiple rules have been applied, only the last one is given.

On the other hand, if NOUN_3 is in the plural number, then it probably takes an "-s" suffix, and for VERB_1, this implies that it has to be in the 3rd person of the singular, and therefore NOUN_1 must be in the singular number. But if NOUN_1 is in the singular number, then it doesn't take the "-s" suffix, and therefore VERB_2 has to be in the plural, and so must be its subject, namely NOUN_2. And NOUN_2 must also be an adjective.

Also, we need that VERB_1 can be used without an object.

To summarize, here are our constraints:

1. we need an adjective that is also a noun in the plural (for ADJ and NOUN_2);
2. we need a noun in the singular that is also a transitive verb in the 3rd person of the plural (for NOUN_1 and VERB_2);
3. and we need an intransitive verb in the 3rd person of the singular, which is also a noun in the plural (for VERB_1 and NOUN_3).

Expected Results

To obtain information on verb valency, we downloaded all English sentences from *tatoeba*[21], a total of 1,800,581 sentences, as of July 16th, 2023, and parsed them with *stanza*.

Here are the results: We found only 31 nouns that are also adjectives and that are not inflected:

> "salmon," "bourgeois," "cod," "Sinhalese," "Nepalese," "Sudanese," "Lebanese," "Vietnamese," "Assamese," "Taiwanese," "Congolese," "Gabonese," "Guyanese," "Togolese," "antiaircraft," "Beninese," "Bhutanese," "Burmese," "Chinese," "Japanese," "Javanese," "Milanese," "Nipponese," "Quebecois," "San Marinese," "Senegalese," "Seychellois," "Siamese," "Singhalese," "Timorese," "Zairese"

This list is quite disappointing because the plurals of "salmon" and "cod" are more frequently "salmons" and "cods" rather than staying uninflected, "bourgeois" is a French word, "antiaircraft" is a bit exotic. All others are nationalities, which means that our perfectly ambiguous sentence runs the risk of being politically incorrect…

As for forms in the other categories, we found 814 for the first category and 978 for the second category. The random sentence we obtained was

> The Siamese serenade bulges.

☞ 404 which makes no sense and is rather an exquisite corpse than a meaningful, perfectly ambiguous sentence.

Finding meaningful sentences among the 24,678,852 combinations is not a trivial task. One could reduce the number of combinations by looking at the cooccurrence frequency of verb and noun pairs. We leave this as an exercise to the reader.

Here are two examples of English perfectly ambiguous sentences among the possible combinations. The examples lack the wit of the French examples but satisfy our criteria and make sense (even though they are not as funny as we expected them to be):

> The Sudanese clone ducks.
> The Vietnamese advocate drinks.

[21] https://tatoeba.org/fr/downloads

Hints for Exercise 4-5: Find emoji that behave like noun phrases

Look at the data in 🗋`comments-with-emoji.txt`. You will notice that the messages are very noisy and in several languages (in fact, we have kept only messages that are mainly English, Spanish, Catalan, Portuguese, and Italian). By parsing them directly, the reader will notice that parsers act randomly when encountering emoji.

A naive method to circumvent this problem is to replace emoji phrases with a common noun (in the language of the message) and then see how it is parsed.

To simplify, we will ignore all emoji at the end of sentences or after punctuation, such as exclamation marks, interrogation marks, and periods. Also, we will search only for emoji displaying objects that can be used as noun phrases. To choose the appropriate emoji, you can consult the Web site SγM名L[22] (pronounced “SYMBL”).

We need to detect its language to use the appropriate *spacy* model and the appropriate common replacement noun for each message. For that, we use the package *langdetect*.

Expected Results

We found the 17 following cases (the first column is the ID of the sentence, and the second column is the language that has been identified):

258	en	I love my 🐕 so MUCH!!!
418	es	Que buena manera de acabar el viernes, pizzas, chuchea y muchas 🍒, Moni, Eli pa comeros
847	es	Se aprovechan en pintura que eres novato y te ponen a quitar y poner bastidores y eso no debería ser que al fin y al cabo los que se llevan los 💰💵 son ellos
1069	en	Another wonderful all hands meeting in the books.. like COMMUNICATION is 🔑 ....
1316	es	Vive en una 🍍 debajo del 🌊
1405	pt	A minha sobrinha comprou um🦜E É NOJENTO!
1797	es	Mejor coordinación entre áreas, nuevas features que nos diferencien positivamente de la competencia y por supuesto: más 💰💰💰 :-)
1816	es	Difícil elección... pero me decanto más por la 🦐 porque la asocio al veranito, las paellas, los chiringuitos...
2853	es	Con Pedro disfrutando su 🥝 ácido
3043	es	Zorpack b.e esta k.o..tenemos un 🚜 y queremos un 🚘
3049	es	Muy contenta...la formadora de cajas de b.e ya no es un🚜...ahora ya tenemos un 🚘...espero que dure😁😁
4123	es	Toda una sorpresa con los🎁🎁🎁que hemos recibido todas. Muchas gracias!😚
4131	es	Demasiados 🚗🚘 en mi carretera... no lo soporto......
4387	es	Felicidades a las cumpleañeras!!! Que tengais muchos🎁🎁 y una gran 🎉🎉🎈🎈🎂🎂
4425	es	🎂🎊🎊🎊🎊🎉🎉🎉por muchos años Xavi que pases un día estupendo y que caigan muchos🎁🎁🎁 un abrazo. Ivón de Vic.
4836	en	COME and see my collection of⚾🥎🏀🏐🏈🏉🎾!!!
5100	es	¡Ha puertas de 🏖🏖🏖y adiós ⌚💻📱🚗👨🏿💼👨🏿🔬🤷‍♂🏼

[22] https://symbl.cc/fr/unicode/blocks/

16.5 Chapter 5: Semantics (and Pragmatics)

Hints for Exercise 5-1: Find faux amis words in French, German, Italian, Spanish, and English

To meet the challenges, we will use the English and the French[23] *WordNet*, as well as a resource called *CogNet* v2[24], and developed by Batsuren *et al.* [5]. From these three resources, we will collect synsets and words representing them. Whenever we find the same word w in language L_1 as a member of synset s_1, and in language L_2 as a member of synset s_2, we will verify that w in language L_1 does not belong to synset s_2 and in language L_2 it does not belong to synset s_1. This is to avoid cases like "port," which means "harbor" in French and "computer port" in English but also has the former meaning in English and the latter in French, so it is not a faux ami.

Once we have gathered these words and the corresponding synsets, we will measure the distance between synsets using the Lesk-Banerjee-Pedersen measure [4]. This is not the only measure between synsets but the only one that worked for us.

Doing this will make us realize that we get far too many results. WordNet has a lot of very specialized synsets for words such as "mesocricetus" (= hamster) or "oestridae" (botfly), which are not really what we want. Therefore, we need a way to restrict ourselves to synsets that actually appear in ordinary texts. For that, we will use a corpus called *SemCor*[25] which is a fragment of the Brown corpus, annotated with WordNet synsets. We will calculate the frequencies of synsets in *SemCor* and keep only those faux amis the synsets of which appear in it.

Be careful to use WordNet v3.0 (not v3.1) since both Wonef (the French WordNet) and CogNet are based on it.

Expected Results

Here are some examples we found by patiently inspecting a list of 2,276 faux-ami candidates:

mare	English: female equine animal	Italian: the sea
gift	English: something acquired without compensation	German: a poison
fine	Italian: the end	English: money extracted as a penalty
mal	German: an instance or single occasion for some event	French: morally objectionable behavior
fame	Italian: hunger	English: the state or quality of being famous
slip	English: a socially awkward or tactless act	French: underpants
muster	English: compulsory military service	German: a sample
four	French: an oven	English: a cardinal number
sale	English: the general activity of selling	Italian: salt
nombre	French: a number	Spanish: a name

[23] https://wonef.pradet.me

[24] https://github.com/kbatsuren/CogNet

[25] http://web.eecs.umich.edu/~mihalcea/downloads/semcor/semcor3.0.tar.gz

Hints for Exercise 5-2: FCA

Defining concepts requires a context (G, M, I), where G are objects, M are features, and I are binary relations. Once we have defined the concepts, objects are distinguished by the values of their features, which become unique signatures for them. These values are binary digits. Think of how many binary digits you need to attribute numbers to five objects uniquely.

As for the second question, do you know of a field that does not have the features of a ring? Or of a ring that does not have the features of a group? And so on.

Expected Results

It is straightforward to find three features for the first question, but necessary to have at least four features for the second question.

Hints for Exercise 5-3: The semantics of Lojban

A tool called *la tersmu* translates Lojban text into multimodal FOL fragments. It can be installed on the user's machine by

```
cabal update && cabal install tersmu
```

or accessed online by an SSH connection

```
ssh tersmu@d.thegonz.net
```

with password `tersmu`.

The name "tersmu"$_{\text{LOJ}}$ is a lujvo built out of "te"$_{\text{LOJ}}$ (a cmavo switching 1st and 3rd places of the selbri following it), and "smuni"$_{\text{LOJ}}$ $\lfloor x_1$ is the meaning of x_2 recognized by $x_3\rfloor$. By switching the 1st and 3rd argument, we get "tersmu"$_{\text{LOJ}}$ $\lfloor$(agent) x_1 understands the meaning of x_2 as being $x_3\rfloor$.

Quoting its author, Martin Bays,

> A panoply of constructions was bolted onto the FOL core [of Lojban] to increase the expressivity to something like that of natural languages. Some of these correspond directly to known formal logics with precise semantics—for example, there are second-order quantifiers using "bu'a," and fragments of the tense system would probably translate directly to temporal logics—but most are far from being formally specified.
>
> All *tersmu* does is to reduce phrases of uncertain meaning in one language (Lojban) to formulae of uncertain meaning in another, which is minimal for the purpose but still rather baroque. I think it's an interesting feature of Lojban that it is possible to do this unambiguously, but it bottoms out way above any standard logic.

This means that the fragments of FOL we will get for your text will probably not be perfect or complete, and the formalism used is not FOL.

Nevertheless, the fragments provided by *tersmu* are an excellent way to discover the semantics of Lojban.

Use *tersmu* on the Lojban sentence and try to understand the meaning of each formula, comparing it with the syntax tree of Exercise 4.3.

Expected Results

In FOL notation, the *tersmu* output is:

$$\begin{aligned} &\mathsf{du'u}\big(\neg\mathsf{ka'e}(\mathsf{se\,cau}_{C_3}(<\mathsf{katna\,vimcu}>(\_,C_1)))\big)(C_4)\\ &\quad\wedge \mathsf{nu}\big(\exists X_1\ (\mathsf{xadni}(X_1)\wedge \mathsf{ka'e}(<\mathsf{katna\,vimcu}>(\_,C_2,X_1)))\big)(C_3)\\ &\quad\wedge \mathsf{stedu}(C_2)\wedge\mathsf{stedu}(C_1)\wedge\mathsf{selmi'ecatra}(C_0)\wedge\mathsf{seldau}(C_4,C_0).\end{aligned}$$

This formalism requires some explanations. First of all, symbols du'u, ka'e, se cau, and nu ("what follows is a statement," "it is possible," "without," "what follows is a situation") are *modal operators*, like the K_A operator we encountered in epistemic logic. As modal operators, they are applied to formulas and produce new formulas. Just like K_A has an argument (the "agent that knows"), Lojban modal operators also can carry arguments, like $\mathsf{se\,cau}_{C_3}$ (in this case, $\mathsf{se\,cau}_A(B)$ means "B without A"). Symbols C_i are constants, and X_1 is a variable.

The notation <katna vimcu> is purely linguistic: it shows that there are two selbri, "katna" and "vimcu," which are combined in a tanru, so that they share the 1st sumti. The tanru inherits the sumti of the second selbri. From a logical point of view, it is just a predicate like any other. Finally, the notation "_" means that that argument is not used.

Once one gets used to the notation, the semantics can easily be deciphered.

16.6 Chapter 6: Controlled Natural Languages

Hints for Exercise 6-1: Discovery of Attempto Controlled English

Download APE (Attempto Parsing Engine) from GitHub[26]. Read the four ACE-related pages that are listed on the top of the page "Attempto Tools and Resources"[27], and especially the "Troubleshooting Guide."

☞ 307 Look at file `clex_lexicon.pl` inside `APE-master/prolog/lexicon`?. It is a Prolog file, so think "unification"!

Expected Results

A high-level perpetrator masters the art of the knocking of an object off a shelf and a table with a precision-timing.
A sudden-leaping expert demonstrates an extraordinary ability and launches himself onto some unsuspecting humans from a hidden corner.
A yarn-connoisseur possesses a keen taste of the unraveling of a ball of a yarn into some intricate and bewildering patterns.

For these sentences to be accepted, a personal dictionary file has to be written, containing the POS tags of several words, in Prolog syntax (similarly to file `clex_lexicon.pl`).

[26] https://github.com/Attempto/APE/releases

[27] http://attempto.ifi.uzh.ch/site/resources/

Hints for Exercise 6-2: How simple is Simple English Wikipedia?

The corpora of a thousand pages are `standard-pages.txt` and `simple-pages.txt`.

We will investigate five criteria:

1. the average syntactic tree length (how long are sentences?),
2. the average syntactic tree depth (how complex are sentences?),
3. the average root degree (how many complements does the root verb have?),
4. the density of proper nouns (how many proper nouns, normalized by page length?),
5. the average difficulty of words (how difficult are words?).

For criteria (1)–(4) a parse with *spacy* will do. Criterion (5) is a bit more tricky. We would be tempted to use the frequency lists of Exercise 1-1, but the difficulty can also be quantified by the age at which a word is learned.

There is a (very recent) corpus that reflects the educational levels to which correspond words: it is called "CLEAR"[28] (CommonLit Ease of Readability Corpus) [9, 10]. This corpus contains 5,000 paragraphs of text to which several metrics have been assigned. We will use the following two: the Bradley Terry text easiness/readability score and the Lexile reading score band. These are assigned to paragraphs. We can consider the minimum paragraph difficulty for each word to induce a difficulty score for words. And for those words in the frequency list but not in the CLEAR corpus, we can still align the two resources to obtain new difficulty values.

Expected Results

Here are the results:

	Standard English W.	Simple English W.
Avg length syntax tree	12.14	10.84
Avg depth syntax tree	2.97	2.69
Root degree	6.20	5.72
Proper noun dens.	0.30	0.38
Avg. word difficulty	0.63	0.66

As the reader can see, the criteria based on syntax trees seem to corroborate our hypothesis: the syntax trees of standard English Wikipedia are indeed longer and deeper, and their roots have (slightly) more neighbors.

Hints for Exercise 6-3: Write haikus inspired by themes by Emily Dickinson in Python

There are many challenges in this exercise. Here are just a few of them:

1. partition Emily Dickinson's poems (or, at least, the 446 poems available on the Gutenberg Project) into thematic clusters, fine enough so that the elements we will take from each cluster will have some coherence and large enough for us to have sufficient elements for our combinations;

[28] https://github.com/scrosseye/CLEAR-Corpus

2. to remain within the limits of an exercise, we will simply recombine elements from her poems. The elements we will choose will be NPs and VPs;
3. we want to create haiku. These have a specific syllabic pattern: 5/7/5. We need a way of counting the number of syllables of any potential haiku verse and sufficient combinations to be sure we obtain results with the required number of syllables.

We will solve challenge #1 by building a graph of dependencies between verbs and nouns in the Dickinsonian corpus. Then, we will filter the graph by frequency and extract communities. We will explore the variation of the number of communities as a function of the frequency filter.

Once we have partitioned the corpus of poems, we will parse them for constituents and for dependencies. We will use the former to build our haiku verses and the latter to gather information about the words to ensure compatibility between the constituents (e.g., number, person, etc.). This approach is naive since all verses will have the same NP+VP pattern, but it is a good start.

Our first attempt will be a random choice of category and a random choice of NP+VP combinations under the haiku syllabic constraint.

A second attempt will be transformer-based similarity for choosing the NP+VP combinations. We will compare the two.

Finally, we will also present haiku generated by ChatGPT and excerpts from "haiku Emily!" [12], a collection of haiku written by a (human) author and inspired by the poems of Emily Dickinson.

Expected Results

Here are some of the results we got:

Many cannot say,
a night ate the fellow, raw,
that earth died for truth.

The flowers flying,
an August day thirsting is,
itself burn away.

Duty speaks to me,
the dew is next to mine,
the trundle bed feels.

A Christian land crawl,
the blond assassin nearing,
everything tasted.

A wing is the noon,
a gayer scarf forget thee,
the grave to bind him.

They are indeed haiku, but the miracle we were waiting for didn't happen. We can hardly consider them as being poetic. Not to mention the problems of grammar.

Our second attempt involves similarity measures using the *sentence_transformers* package. For that, we calculate the similarity of all candidate pairs (there are between 10k and 34k candidate pairs, depending on the community), a calculation of 436,995 similarities that took 11 hours on a MacBook Pro 2,3 GHz Intel Core i9 8 cores with 32 MB of RAM. The code is essentially the same, except that instead of choosing combinations of constituents randomly, we take those of maximum similarity. Here are some of the results:

No universe wore,
the meek members strains and spins,
a flower will do.

The motions flitting,
flame denote that fire was,
Triumph overcome.

Loneliness thirsting,
a boundless place dwelling there,
such permitted so.

Agony was pain,
immortality is death,
agony aching.

Anguish suffering,
soft is what I cannot reach,
shiver trembling

Obviously, the verses are thematically more consistent, but the result is still not very convincing. It is a start, though, and we leave it as an open exercise to the reader to improve the results.

For the sake of comparison, here are haiku generated by ChatGPT 3.5 when asked to write "haiku poems in the style of, and using themes by, Emily Dickinson":

Hope, a fragile bird,
Sings in the heart's quiet depths,
Unseen, but felt near.

Nature's mystic dance,
Whispers secrets to my soul,
Through each leaf and glance.

A slant of sunlight,
Touches the soul's hidden depths,
Awakening life.

In silence I dwell,
A world of thoughts unfurling,
Vast, uncharted realm.

A bloom in stillness,
Unfurls its petals of grace,
Nature's gentle hand.

And here are five haiku written by Everett Decker [12][29] and based on specific poems by Emily Dickinson:

I draw the curtains
a ruffle of feathers
blast

my parting
unlike the maple's
is silent

our spirit —
a bird that dances
in its flight

[29] https://haikuemily.weebly.com/about-haiku-emily.html

pond lilies
cradle
the still child

his voice still
his echo
still sounding

Hints for Exercise 6-4: Do Daleks use a controlled language?

Daleks appear only in 94 of the 630 Dr Who episodes. They have 2,558 speech turns. In Fig. 16.8, the reader can see the number of sentences per episode over the entire 1963–2022 period.

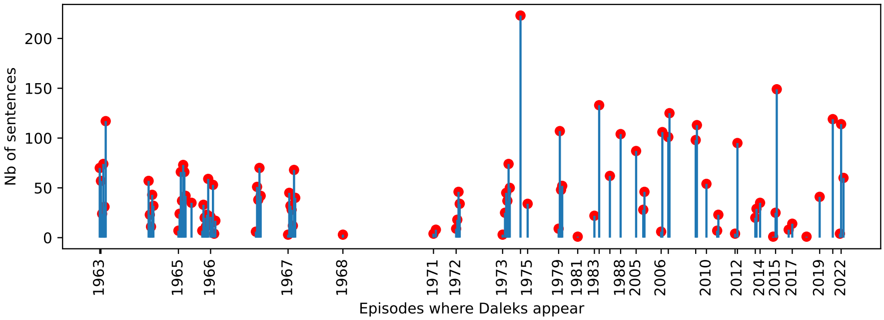


Fig. 16.8 Number of Dalek sentences per episode, in the period 1963–2022

The Dalek corpus is provided as file `daleks.txt`, together with episode numbers. In the file, we have segmented speech turns into sentences, providing 4,256 sentences.

We will parse the 4,256 with *stanza* to obtain constituents. Then, we will extract the number of production rules necessary for each syntax tree. We will keep only *n*-ary rules with an *n* greater than 1 (unless the left part of the rule is the root). Once we have obtained the rules necessary for each utterance, we will study the complexity of the rule sets and calculate the number of utterances covered by the most frequent production rules.

We will then randomly choose the same number of utterances among the ten companions with the most speech turns (in order of decreasing frequency: Clara, Jamie, Ian, Sarah, Romana, Brigadier, Barbara, Jo, Amy, Rose). The file is called `companions.txt`. We will apply the same calculations and compare the results.

Expected Results

We obtain the following results:

	Daleks	Companions
Distinct POS tags	38	41
Distinct lemmas	1,422	1,714
Avg rank of words	6,139	5,767
Distinct syntax tags	61	61
Distinct syntax rules	720	1,377
Avg nb rules per sentence	3.31	3.89
Avg size left rule side	1.74	2.01

The number of distinct lemmas shows that companions have a larger vocabulary, and the almost double number of syntax rules shows that companions have a more flexible syntax. The ten more frequent non-stop lemmas for the two corpora are:

Dalek: 'exterminate,' 'obey,' 'Doctor,' 'destroy,' 'move,' 'time,' 'prisoner,' 'locate,' 'report,' 'control.'

Companions: 'Doctor,' 'go,' 'get,' 'know,' 'look,' 'come,' 'think,' 'see,' 'say,' 'right.'

There is no surprise in these two lists: the first summarizes the Daleks' essence, while the second is a regular list of human activities. The word 'doctor' is the most frequent because the favorite activity of companions is to address the Doctor.

In Fig. 16.9, the reader can see the coverage of the 140 most frequent syntax rules, in the sense that a set of production rules covers a sentence if its syntax tree can be built using the given rules. As the reader can see, the difference is striking: with only 140 production rules,

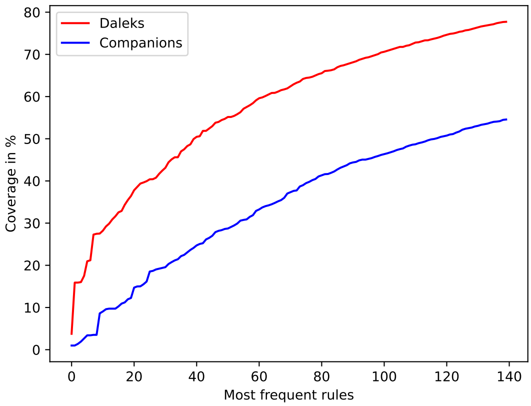


Fig. 16.9 Coverage of the 140 most frequent syntax rules for the Daleks corpus and the 140 most frequent syntax rules for the Companions corpus

one can parse almost 80% of the Dalek corpus but only about 50% of the Companions corpus.

We will attempt to build a *Controlled Dalek Language* (CDL) using exactly these 140 rules. As can be observed from the Companion corpus, a text of this size requires about 1,377 different production rules. Requiring for roughly a tenth of the regular number of production rules is a very strong constraint.

We have identified the sentences of the Dalek corpus that belong to CDL. Fig. 16.10 is a copy of Fig. 16.8 on p. 482, to which we added green strokes for sentences in CDL. As the reader can see, when Daleks do not utter many sentences, the ratio of CDL is higher, probably

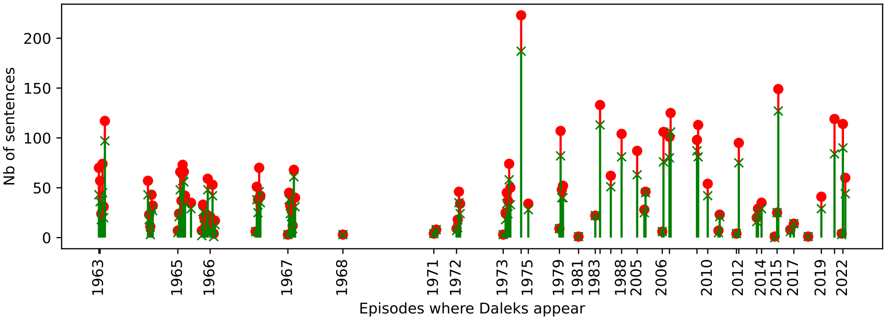


Fig. 16.10 In green: number of sentences in CDL per episode; in red: number of sentences that cannot be parsed with the CDL rules

because in those episodes, Daleks have only minor parts (uttering the usual "Exterminate!" and hardly anything more complex), while in episodes where Daleks play a more important, sentences requiring production rules outside the controlled ones are more frequent.

To conclude this exercise on controlled languages, here is some additional data on the Dalek language. The repetition rate is very high, here are the sentences that appear more than 20 times in the corpus:

Sentence	Frequency
Exterminate!	328
I obey.	62
Emergency!	51
We obey.	27
Alert!	26
Move!	23
Yes.	23

As Daleks appear in 94 episodes, it results that they utter approx. 3.5 "Exterminate!" per episode in which they appear, which is less than one would expect. The frequent sentences all belong to CDL. A last piece of information: the outlier episode in 1974, containing a record of 223 (!) sentences uttered by Daleks, is the episode *Death to the Daleks* S11E3, with the 3rd Doctor and Sarah Jane Smith as companion. It aired on February 23rd, 1974.

16.7 Chapter 7: Graphs

Hints for Exercise 7-1: Using WordNet for disambiguation

☞ 143 The structure of WordNet is as follows: a *word* can have one or more *senses*; senses are gathered in *synsets*. Between synsets, one may have relations such as hyperonymy, meronymy, etc. Nouns, verbs, and adjectives belong to different resources, even though relations may occur between different parts of speech. Together with each synset is provided a short textual information, called a *gloss*, which again is subdivided into a *definition* and one or more *examples* (between double quotes).

To illustrate WordNet's structure with an example, the word "bank," as a noun, has ten senses:

- bank#1 (sloping land (especially the slope beside a body of water)) "they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"
- depository financial institution#1, bank#2, banking concern#1, banking company#1 (a financial institution that accepts deposits and channels the money into lending activities) "he cashed a check at the bank"; "that bank holds the mortgage on my home"
- bank#3 (a long ridge or pile) "a huge bank of earth"
- bank#4 (an arrangement of similar objects in a row or in tiers) "he operated a bank of switches"
- bank#5 (a supply or stock held in reserve for future use (especially in emergencies))
- bank#6 (the funds held by a gambling house or the dealer in some gambling games) "he tried to break the bank at Monte Carlo"
- bank#7, cant#2, camber#2 (a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force)
- savings bank#2, coin bank#1, money box#1, bank#8 (a container (usually with a slot in the top) for keeping money at home) "the coin bank was empty"
- bank#9, bank building#1 (a building in which the business of banking transacted) "the bank is on the corner of Nassau and Witherspoon"
- bank#10 (a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)) "the plane went into a steep bank"

In this list, #*n* denotes the sense's number, and glosses are represented between parentheses.

We will use a very naive technique for disambiguating the word "bank" in our corpus: we will create a graph connecting

1. words with senses,
2. senses with synsets,
3. synsets with other synsets;
4. and in a second phase, we'll try to go further and connect words from glosses with senses.

This graph will contain the complete structure of WordNet.

Then, to classify each sentence as belonging to the class "financial institution" or the class "terrain alongside a river," we will take each one of its nouns, and we will calculate the shortest path between it and the word "bank." As words are connected with the graph only through senses, this path necessarily traverses one of the word "bank" 's senses, and this will be a candidate for the class of the sentence. By a voting process, we choose the best candidate and assign its class to the sentence.

Since some of the senses are close to one of the senses we are looking for, we will simplify the situation by declaring that senses #1, #3, #7, and #10 belong to the class "geography" and senses #2, #6, #8, and #9 belong to the class "finance." When senses #4 and #5 are encountered, we will vote randomly.

Expected Results

In our first attempt we get a graph of 492,377 vertices and 493,295 edges with the following performance:

	precision	recall	F-score
`geo`	0.9032	0.5600	0.6914
`fin`	0.6812	0.9400	0.7899

In a second attempt we parse glosses using *TreeTagger* but keep the same nodes and increase the number of edges only. We get a graph with 1,093,042 edges, a 122% increase with respect to the previous setting. Here is its performance:

	precision	recall	F-score
`geo`	0.6049	0.9800	0.7481
`fin`	0.9474	0.3600	0.5217

Comment the results and look at the actual paths to understand what has happened

Hints for Exercise 7-2: Find the most central word of the Quran and of the Bible

The texts can be freely downloaded from the Web: we chose the translation of the Quran by Talal Itani[30] and a translation of the Bible in Basic English[31]. A very efficient solution for extracting text from PDF files is to convert them first to Microsoft Word format using Adobe Acrobat Pro and then save the Word file in raw text format. We will use *stanza* to parse the files.

As for building the graph, this is just a toy example, so that we will do it as simply as possible: we will mark the (proper) nouns connected to a verb in each sentence. If there are at least two nouns connected with the same verb, we will create nodes for the nouns and add an edge between each pair of them.

This method has many drawbacks:

1. it misses entirely all coreferences: we will not attempt anaphora resolution so that a sentence like

   > He was a very great bowman (Gen 10:9)

   will not contribute to the graph;
2. verbs with a single noun linked to them, as in

   > And God said (Gen 1:3)

   do not contribute to the graph, since *stanza* considers "And God said" as a complete sentence, and there is no second noun in it;
3. lists are lost, for example, in

   > And these are unclean to you among things which go low down on the earth; the weasel and the mouse and the great lizard, and animals of that sort; (Lev 11:29)

   there is only one verb, namely "go." There is no noun connected to it: "things" is connected to "unclean" (the root of the sentence), "earth" is connected to "down," "weasel" is connected to "things," "mouse," "lizard," and "animals" are connected to "weasel," and "sort" is connected to "animals." In our simplistic setting, none of these nouns contributes to the graph.

---

[30] `https://www.holybooks.com/wp-content/uploads/quran-in-modern-english.pdf`

[31] `https://www.o-bible.com/bbe.html`

A particular feature of our approach is that it does not take frequency into account: no matter how frequent a noun is, it will contribute with a single node. But if it is frequent, chances are it is connected with many other nodes so that it will become central indirectly by connecting with other nodes.

Expected Results

The results are as follows: both graphs have a giant component (1,934 vertices out of 1,951 for the Bible and 1,059 vertices out of 1,124 for the Quran). The twenty most central nouns of the Bible are:

> man, Lord, day, place, king, people, time, God, land, way, hand, death, child, word, son, house, servant, end, earth, father

and for the Quran:

> God, Lord, people, day, him, messenger, Moses, sign, night, truth, home, none, son, forgiveness, revelation, heart, believer, news, knowledge, punishment

There are five common nouns: "God," "Lord," "people," "son," and "day."

Hints for Exercise 7-3: Assortativity of Chinese Hyperonyms

We will use the files `IDS-UCS-Basic.txt` and `IDS-UCS-Ext-B.txt`, which we used already in Exercise 2-4. The first problem is that the decomposition of sinograms in these files is not *exhaustive*, in the sense that when a sinogram is decomposed as AB, the parts A and/or B may very well have their own decompositions. To solve this problem, we will create a new resource that will decompose every sinogram until it is impossible to go any further.

The Chinese WordNet (also known as COW) [29] can be downloaded[32] from the site of the Open Multilingual WordNet [6]. It contains 80,009 words (built out of 4,531 distinct sinograms), belonging to 42,228 synsets.

For each word in the COW, we will concatenate the decomposition of its characters into a single list, which we will call the "feature set" of the word.

We will collect all hyperonym relations in COW. Each is a relation between synsets so that we may have many words in the source and many words in the target of the hyperonym relation. We will compare all pairs of words between source and target and keep the one with the highest similarity. We will measure similarity between feature sets as twice the number of common elements (taking repetitions also into consideration, e.g., $[a, a, b]$ and $[a, a, c]$ have *two* common elements) divided by the sum of the lengths of the two sets.

This calculation will give us a number, but how do we know if the relation is assortative?

One way of estimating assortativity is by comparing the result with the result of the same operations made on random (and hence not necessarily hyperonymic) word pairs. Therefore, for each hyperonymic relation, we will take two random words with the same length as the original ones and calculate their similarity. We will then divide the average real similarity by the average random similarity. If the quotient is greater than 1, we have an assortative phenomenon. If it is less than 1 we have a disassortative phenomenon, and if it is close to 1, then there is no correlation between features and hyperonymy.

[32] https://github.com/omwn/omw-data/blob/main/wns/cow/wn-data-cmn.tab

Expected Results

The result is 4.1187, meaning that *the hyperonymy relation is strongly assortative* since we have four times as many common features between hyponym and hyperonym than between two arbitrary words of the same lengths.

Hints for Exercise 7-4: Productivity of sinographic component base

Since this exercise belongs to the chapter on graphs, the reader may already have guessed that the solution is graph-theoretic. Indeed, we can build a graph, the vertices of which are sinograms, and the edges represent the fact that a sinogram is a component of another sinogram. This graph will be bipartite, the two partitions being primitive and decomposable sinograms. And what we described as the base set and the production set will be the two partitions of a connected component.

Taking a connected component guarantees that every decomposable sinogram is connected to all its components and that the components do not participate in any other decomposable sinogram.

We will, therefore, look for connected components of the graph built out of the sets that are requested. We have extracted the set of Jōyō kanji from Wikipedia and have stored it in file 🗋 `joyo-kanji.txt`, together with the grade where it is introduced. As for the other sets, we will use the decompositions of Unicode sinograms from Exercises 2-4 and 7-3.

Imagine these graphs for a moment. How many connected components do you expect to find?

Expected Results

The results are astonishing. For the first question, the basic "CJK Unified Ideographs" table, which contains 20,992 sinograms, one would expect a swarm of small components creeping around. That is not at all what happens. In fact, we have only *two* (!) components, a large one (4,336 sinograms, out of which 386 are primitives) and a tiny one (one primitive: 旡, one decomposable: 兓). There is no mistake for the tiny one since the decomposable is exactly twice the primitive sinogram.

What happens with the remaining 16,654 sinograms in the table? Well, they are indeed decomposable, but their decomposition contains at least one component outside the "CJK Unified Ideographs" table. The 4,338 we obtained are, somehow, the ones with a "pedigree," where we know that all of their components belong to this set.

What is surprising is that there are no other components. This shows that the primitive characters are very efficient and combine in all possible ways. It may also be that the Unicode table has been engineered so that there is only one large component (and 旡 somehow escaped from this logic).

When we add the table of "CJK Unified Ideographs Extension A" to our set (a total of 27,584 characters), we get again a giant component (5,453 sinograms, our of which 390 are primitive) and the same tiny exception as previously.

When we add the table of "CJK Unified Ideographs Extension B," which is much larger, we attain a total number of 70,304 ideographs. Once again, we have a giant component (10,161 sinograms, 421 of which are primitives), and this time not one but two tiny exceptions: the one we inherited from the previous case, and a new one: 飞 as a unique component of character containing it twice.

Interestingly, the size of the giant component increases more significantly than the number of primitive characters, which evolved from 386 to 390 to 421, while the size of the component almost doubled.

Let us now consider the case of the Jōyō kanji. In the following table, we summarize the evolution of the situation from 1st to 6th grade to the complete set:

grade (cumul.)	1st	2nd	3rd	4th	5th	6th	High school	Unicode
total	80	240	440	642	835	1,026	2,136	20,992
nb. components	7	15	14	12	11	6	2	2
size of giant c.	7	84	192	287	362	454	769	4,336
ratio giant c. (in %)	8.75	35	43.64	44.70	43.35	42.75	36	20.66
prim. of giant c.	4	41	90	127	154	186	246	386
ratio of prim. (in %)	57.14	48.81	46.88	44.25	42.54	40.97	31.99	8.9
avg. size of other c.	4.17	4.14	4.69	3.91	3.8	3	3	1
ratio giant/second c.	1.173	7.64	11.29	41	40.22	151.33	256.33	4,336

As we see, the "giant" component starts by being comparable to the other components and slowly becomes giant: in the 1st grade, the largest component is almost of the same size as the second largest, while for the complete Jōyō set we have a ratio of 256 between them. The ratio of the giant component increases and then decreases again, with a high point at 44.7%, while the ratio of primitives of the giant component steadily decreases, from 57.14% to 31.99%. In the last column, we have included the values of the Unicode "CJK Unified Ideographs" table for comparison.

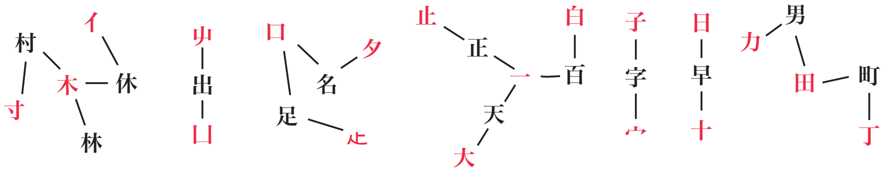


Fig. 16.11 The connected components of 1st grade Jōyō kanji, primitive sinograms are displayed in red

In Fig. 16.11, the reader can see the 1st-grade graph, 32 sinograms among the 80 taught in first grade. They are divided into seven connected components. We have displayed primitive sinograms in red.

16.8 Chapter 8: Formal Languages

Hints for Exercise 8-1: a^n, $a^n b^n$, $a^n b^n c^n$, etc.

$L_1 = \{a^n \mid n \in \mathbb{N}\}$ is regular, you need one rule to add a letters and one rule to stop.

$L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$ is context-free. You need one rule to add ab pairs in the center of the word and one rule to stop.

$L_3 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is context-sensitive. You first produce as many aBC as necessary (lowercase letters being terminal and uppercase letters intermediate symbols), then find a way to replace bCB by bbC from the center to the right boundary.

Hints for Exercise 8-2: How many formal languages can there be?

We need to do the following calculations. If A is the size of the alphabet, M is the maximum number of letters in a word, and N is the maximum number of words in the language, then

1. how many words of size m are there?
2. how many words of size between 1 and M are there?
3. how many languages contain n words?
4. how many languages contain less than N words?

As for the size of the universe, according to Wikipedia[33] the volume of the observable universe is around 3.566×10^{80} m^3, and according to `homework.study.com` there are 8.49×10^{28} atoms of iron per cubic meter. Therefore, if the observable universe were a giant iron ball, it would contain 3.028×10^{109} iron atoms.

Expected Results

In scientific notation, the number of possible languages of less than 3,000 words of less than ten letters is approx.

$$4.813 \times 10^{33,369},$$

that is a number with 33,370 digits. Compared with this number, the number of atoms of an iron-made universe, namely 3.028×10^{109}, is peanuts.

Hints for Exercise 8-3: The complementary of a regular language

Every regular language can be represented by a deterministic complete FSA. What would be the FSA of the complementary of a language?

Hints for Exercise 8-4: Implement French and German gender-neutral language

Concerning gender-neutral forms, our main reference will be [17].

For French we will use a wonderful resource: *Lefff* ("Lexique des formes fléchies du français"$_{\text{FR}}$ ⌊Lexicon of French inflected forms⌋) [26][34]. Take for example the first noun of our challenge sentences: "boulanger"$_{\text{FR}}$ ⌊baker⌋. In Lefff we will find the following four lines:

---

[33] https://en.wikipedia.org/wiki/Observable_universe

[34] http://pauillac.inria.fr/~sagot/index.html#lefff

```
boulanger 100 nc [pred="boulanger_____1 ... @ms] ...
boulangers 100 nc [pred="boulanger_____1 ... @mp] ...
boulangère 100 nc [pred="boulanger_____1 ... @fs] ...
boulangères 100 nc [pred="boulanger_____1 ... @fp] ...
```

where we have replaced information irrelevant to this task with `...`. What we will extract from these lines is the fact that the base form "boulanger" has four inflected forms: masculin plural "boulangers," feminine singular "boulangère," and feminine plural "boulangères." It belongs, therefore, to Class 8 of [17, p. 67].

In our code, we will gather all non-epicene nouns and adjectives from Lefff, and if we can detect one of the 16 classes of [17, p. 67], we will calculate the gender-neutral form and add it to the dictionary of that lemma.

Then, we will parse each sentence twice with dependency *stanza*. The first time, we will find all nouns that are candidates for a gender-neutral form. If the gender-neutral form exists, we will keep the ID of the noun and parse the sentence again. The second time, whenever there is a word depending on a gender-neutralized word, we will convert it as well.

As for the German part, unfortunately, there is no Lefff for German, or at least the author is unaware of it. We had to extract information from the German Wiktionary.[35] Fortunately, the feminine forms are easy to extract since they systematically can be found in a section called "{{Weibliche Wortformen}}"$_{\text{DE}}$. We extracted 6,867 pairs of masculine/feminine nouns. Out of them, 5,212 belong to Class 1 of [17, §7.2.1], 574 to Class 2, 72 to Class 3, 467 belonged to a class where the masculine form ends with "-er" and the feminine form with "-e" (as in "Abgeordneter/Abgeordnete"). We also found 154 cases of Class 1 where the feminine stem is umlauted (as in "Arzt/Ärztin") and 24 cases of Class 2 (as in "Jude/Jüdin").

We will use the code of the French part, adapted to German.

16.9 Chapter 9: Logic

Hints for Exercise 9-1: Socrates' "I know that I know nothing"

An approximation of "I know that I know nothing" in FOL would be "I know one thing, and I know nothing." Show that this formula is a contradiction.

Epistemic Logic uses the knowledge operator K_S (where S is Socrates), but Epistemic Logic is based on propositional logic and therefore has no quantifiers. An approximation of "I know that I know nothing" would be "Let φ be an arbitrary formula, then $K_S(\neg K_S\varphi$. Find the S5 axioms of Epistemic Logic and show that this formula contradicts them.

The saying "I know that I know nothing," as well as its Greek version "ἓν οἶδα ὅτι οὐδὲν οἶδα"$_{\text{EL}}$ ⌊I know one thing, namely that I know nothing⌋ appears nowhere in the works of Plato, the biographer of Socrates. Instead, Socrates has been quoted in his *Apology* (21d8) to have said "ἃ μὴ οἶδα οὐδὲ οἴομαι εἰδέναι"$_{\text{EL}}$ ⌊what I do not know, neither do I think I know⌋, as translates Fine [13, p. 39] in his elaborate essay about the "I know that I know nothing" saying.

This, probably much more authentic, sentence by Socrates uses two verbs: "to know" (both affirmatively and negatively) and "to think" (negatively). We could strengthen the argument a bit by replacing "I don't think I know" with "I know I don't know," obtaining thereby the formula "for an arbitrary φ, $\neg K_S\varphi \rightarrow K_S(\neg K_S\varphi)$. Check whether this formula is compatible with S5 axioms of Epistemic Logic and, therefore, not a contradiction.

[35] https://dumps.wikimedia.org/backup-index.html

Hints for Exercise 9-2: Use Prolog for Formal Semantics

The aim of this exercise is not to teach the reader Prolog but to give her a brief overview that might encourage her to become more involved later on.

At least for those used in imperative or object-oriented programming languages like Python or C, Prolog's approach to programming is commonly called an "epistemological obstacle": it makes reading Prolog programs difficult and can block the learning curve.

The key to understanding Prolog is the notion of *unification*. In FOL, unification consists of making two formulas "look-alike" by making *substitutions*. A substitution replaces a variable with a term. ☞ 278 For example, take formulas $P(f(X), a)$ and $P(f(b), Y)$, where, following the usual conventions, P is a predicate symbol, f a function, X and Y are variables, and a and b are constants. We obtain the same formula if we substitute X by b (a constant is a term) and Y by a (these substitutions are written X/b and Y/a). These two substitutions constitute a unification of the formulas.

Prolog does nothing else than unify formulas. Let us take an example. Write the line

```
1 companion(rory).
2 companion(amy).
3 loves(rory,amy).
4 mother(amy,melody).
5 parent(X,Y) :- mother(X,Y).
6 s([rory, loves, amy]).
```

(don't forget the points) into a file called `kb.pl`. Lines 1–4 and 6 will be our initial *facts*.

After installing Prolog[36], open a terminal, go to the directory containing `kb.pl` and launch Prolog by typing `swipl`. A `?-` prompt will appear. By typing

```
[kb].
```

the facts contained in the file are read. By typing[37]

```
loves(X,Y).
```

Prolog replies:

```
X = rory,
Y = amy.
```

This apparently innocuous answer must be interpreted correctly: the equals sign denotes *substitution* (and not equality!), the comma denotes *conjunction* (like $\wedge$ in FOL), and the period is the end of the statement. What Prolog is saying is:

> I attempted to unify the formula you gave with the formulas in the knowledge base contained in `kb.pl`, and my attempt was successful: the (unique) unification I found consists of substitutions X/rory and Y/amy.

And if we ask

```
companion(X).
```

we will first get the answer

```
X = rory
```

[36] See https://www.swi-prolog.org/download/stable.

[37] Notice that constants and predicates start with a lowercase letter and variables with an uppercase letter.

without a point or a prompt, there are other answers. By typing `&` (which is in fact a disjunction[38]), Prolog will answer

```
X = amy.
```

this time is followed by a point and prompt, meaning that these are all our substitution choices.

Line 5 of file `kb.pl` is a *rule*. Imagine `:-` is a $\longleftarrow$. In FOL the rule would be

$$\forall X, Y\, \mathsf{parent}(X, Y) \longleftarrow \mathsf{mother}(X, Y),$$

i.e., every mother is a parent. We can test Prolog's inference capacity by asking

```
parent(amy,melody).
```

and, like any proud Whovian, Prolog will answer with a conspicuous

```
true.
```

Understanding the role of unification in Prolog was our first epistemological hurdle. The second one is the use of *lists*.

The notation `s([rory, loves, amy])` on line 6 of file `kb.pl` is, in fact, a list of three constants given as an argument of a predicate `s`. We can unify it with

```
s([X,Y,Z]).
```

obtaining as answer,

```
X = rory,
Y = loves,
Z = amy.
```

but this would only work with a list of exactly three elements. Prolog provides the notation `[X|Y]` to deal with lists of arbitrary elements. This notation is misleading because of its symmetric look: it is *not* symmetric. `X` will be unified with the first element of some list, and `Y` with the remainder. In other words, `X` is an element, and `Y` is a list. If we ask for

```
s([X | Y]).
```

we will get the answer

```
X = rory,
Y = [loves, amy].
```

and if we ask for

```
s([X | [Y | Z]]).
```

thereby using the notation twice, we will get

```
X = rory,
Y = loves,
Z = [amy].
```

[38] These are the vagaries of programming language notations: `&` is a *disjunction* for Prolog and a (bitwise) *conjunction* for Python, c'est la vie…

Observe that `Z` is a list, admittedly a list of a single element, but a list nonetheless. Many problems, including the problem of describing the syntax and semantics of natural language sentences, are solved by using this tool and, therefore, by cutting lists into *head* (first element) and *tail* (the remaining elements).

To solve the exercise read § 9.1–9.3, 9.5 and 9.7 of [23] or online tutorials,[39] or go directly to the solution.

Expected Results

In the hints for this exercise, we presented two of Prolog's epistemological obstacles: unification and lists. Here comes the third one: *difference lists*. First of all, let us see the problem that makes this notion necessary.

Imagine we want to write a Prolog rule saying that a sentence can consist of an NP followed by a VP. A first attempt would be

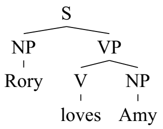
```
s(X) :- np(Y), vp(Z).
```

As we already mentioned, `:-` corresponds to $\leftarrow$ and the comma to $\wedge$ in FOL, so this formula becomes $\forall X, Y, Z\, \texttt{s} \leftarrow \texttt{np} \wedge \texttt{vp}$. But the conjunction is commutative, so when writing this formula, we would automatically also have $\forall X, Y, Z\, \texttt{s} \leftarrow \texttt{vp} \wedge \texttt{np}$, which sounds more like Yoda language than like standard English. This is not good.

If the comma is unable to enforce the order of constituents, we need some other way of doing it. The solution is to use *difference lists*, by writing

```
s(L1,L) :- np(L1,L2), vp(L2, L).
```

By having `L2` as *second* argument of `np` and, at the same time, *first* argument of `vp`, we definitively establish the fact that `np` comes *before* `vp`. To understand how this works, let us take a simplified version of the "Rory loves Amy" tree, with only five non-lexical nodes:



This tree requires the following rules:

S → NP VP
VP → V NP
V → loves
NP → Rory
NP → Amy

In Prolog, they are translated as

```
s(L1,L) :- np(L1,L2), vp(L2,L).
vp(L1,L) :- v(L1,L2), np(L2,L).
v([loves | L], L).
np([rory | L], L).
np([amy | L], L).
```

[39] E.g., https://www.metalevel.at/prolog/dcg, https://www.amzi.com/manuals/amzi/pro/ref_dcg.htm.

Lines 3–5 are indeed mysterious. We will see what they are good for. Let us replace the contents of `kb.pl` with these five lines and ask Prolog:

```
s([rory, loves, amy], []).
```

We will get the answer

```
true.
```

which is logically correct, but psychologically, it was a bit disappointing after all the work we did. Here is a more interesting question:

```
s(L,[]).
```

and we will get the following answer (after typing thrice `&`):

```
L = [rory, loves, rory] ;
L = [rory, loves, amy] ;
L = [amy, loves, rory] ;
L = [amy, loves, amy].
```

that is all possible unifications of `L`, and therefore *all* grammatical sentences that are compatible with the two rules and three facts we gave.

Let us now take Prolog's reasoning apart. Line numbers will refer to the contents of the `kb.pl` file as above. When we ask

```
s([rory, loves, amy], []).
```

Prolog takes the rule of Line 1 and makes the obvious substitution of `L1` by `[rory, loves, amy]` and `L` by `[]` (the empty list). What `L2` can be, we don't know yet. Prolog goes to the fact of line 4 and asks itself: "Can I unify `[rory, loves, amy]` with `[rory | L]`? because if that works, then I will get `L2` which will have to be `L`." But, by the very definition of the notation `[... | ...]`, the unification `[rory, loves, amy]` with `[rory | L]` is possible and amounts to the substitution of `L` with `[loves, amy]`. Going up, we now have the `L2` of line 1: it is `[loves, amy]`. Let us place it inside the `vp` of line 1 to check whether it works. `vp` is defined on line 2. Using line 2, we must have `v([loves, amy], L2)` and need to find what `L2` is. Fortunately, there is line 3 saying that we have the fact `v([lives | L], L)`, by unifying it with `v([loves, amy], L2)` we get that the `L2` of line 2 can only be `[amy]` (a singleton list). We now apply this substitution to the `np` predicate of line 2 and get `np([amy], L)`. Comparing this with the fact of line 5, we get that the `L` of line 2 must be empty. Going up, we conclude that the `L` of line 1 must also be empty, and so is the second argument of `s` on line 1. This means that our unification succeeded since we showed that when the first argument of `s` is `[rory, loves, amy]` then the second must be `[]`, as in our question.

It becomes clear now why this technique is called *difference lists*: intuitively, line 1 says, "the difference between `[rory, loves, amy]` and `[]` is a sentence if the difference between `[rory, loves, amy]` and `[loves, amy]` (this difference is, in fact, `[rory]`) is a NP and the difference between `[loves, amy]` and `[]` is a VP." The same argument is applied on line 2, and we get that the difference between `[loves, amy]` and `[]` is a VP if the difference between `[loves, amy]` and `[amy]` (this difference is in fact `[loves]`) is a verb and the difference between `[amy]` and `[]` is a NP. At that point, we have three unanswered questions: Is `[rory]` (resp. `[loves]`, `[amy]`) an NP (resp. a verb, an NP)? The answers are given by the three facts on lines 3–5, which say exactly that. By this incessant ballet, going downwards and then upwards again, and so forth, Prolog manages to unify our query with the rules and facts of file `kb.pl`.

Using difference lists is nice but not something you want to do all the time since the many `L`, `L1`, `L2`, etc., symbols are only the means, not the end. Therefore, Prolog introduces a special notation: the `-->` operator. Our formal grammar can be rewritten as:

```
s --> np, vp.
vp --> v, np.
v --> [loves].
np --> [rory].
np --> [amy].
```

Rules with this notation are called DCGs (Definite Clause Grammars). They can only be used in the knowledge base part. Queries and answers remain as usual.

In *nltk*, the step after defining context-free grammars (p. 264) was the definition of feature-based grammars (p. 265). In Prolog, the infrastructure is already there: each feature appears as a variable in rules and as a constant in the left part of facts. For example, to have a singular and a plural version of the verb "to love" and to express number agreement with the subject, one would use:

```
1  s --> np(Number), vp(Number).
2  vp(Number) --> v(Number), np(_).
3  v(sing) --> [loves].
4  v(plur) --> [love].
5  np(sing) --> [rory].
6  np(sing) --> [amy].
7  np(plur) --> [people].
```

Notice on line 2 the underscore argument for `np`: as `np` predicates are unary, we need an argument, and `_` denotes the fact that the value can be arbitrary. Adding an argument to predicate symbols in DCG notation amounts to adding an argument in "real" Prolog. E.g., line 1 of the code above is

```
s(L1,L1) :- np(Number,L1,L2), vp(Number,L2,L).
```

in real life.

Remember how we used a special feature ("`sem`") to *nltk* feature-based grammars to carry semantics (p. 172)? We will do the same thing in Prolog. The notation is reminiscent of CCGs (p. 173): in CCGs, intermediate-level categories are programmatic in the sense that they contain actions allowed, as in "(S\NP)/NP" meaning that this node expects first an NP on the right, then an NP on the left, and returns an S. In Prolog, we will call `Semantics` the variable that will contain the semantics of the whole sentence. This argument will be added to all predicate nodes. The value of that argument for the verb will be `Object^Subject^Semantics`, meaning: I first expect an object, then a subject, and I will return the semantics of the sentence.

Here is the new grammar (we have omitted the number argument for the sake of brevity):

```
s(Semantics) --> np(Subject), vp(Subject^Semantics).
vp(Subject^Semantics) --> v(Object^Subject^Semantics), np(Object).
v(Y^X^loves(X,Y)) --> [loves].
np('rory') --> [rory].
np('amy') --> [amy].
```

and asking the question:

```
s(Semantics, [rory, loves, amy], []).
```

we get the answer:

```
Semantics = loves(rory, amy).
```

Isn't that beautiful? How does it work? The notation `X^` is actually the λ-expression λX. To unify `Object^Subject^Semantics` with `Y^X^loves(X,Y)` we apply the three substitutions

`Object/Y`, `Subject/X` and `Semantics/loves(X,Y)`. Then we go down to the lexical level and substitute `X` by `amy` and `Y` by `rory`, and we are done.

We have only scratched the surface of the tip of the iceberg (we just merged two metaphors) of Prolog, and we hope to have convinced the reader of the usefulness and beauty of this language.

Hints for Exercise 9-3: Prove that Curiosity killed the Cat using Attempto Controlled English, *gkc*, and RACE

On using Attempto Controlled English, see Exercise 6-1. Take each of the four sentences of the KB and the negation of the query, and attempt writing them in Attempto (pun!) by adding the appropriate entries to the user lexicon file. Once Attempto's parser accepts your sentences, check the paraphrase to be sure Attempto has captured the desired meaning.

Once the parser accepts the entire set of sentences, convert it to FOL. Write a utility to translate Attempto's FOL output into LaTeX for inspection.

gkc (Graph Knowledge Core) [27] is a prover of theorems written in FOL. It can be installed from GitHub[40]. *gkc* is similar to *Z3*, except that it can read files in the TPTP[41] FOF (First-Order Form) syntax, and this is precisely one of Attempto parser's output formats. By submitting Attempto's TPTP output to *gkc*, we can prove that it was indeed Curiosity who killed the cat.

Finally, submit the Attempto Controlled English sentences to RACE,[42] Attempto's reasoner [14], and obtain the same result through a simple query.

Expected Results

The expected Attempto Controlled English sentences are

1. Jack owns a dog.
2. If somebody X1 owns a dog then if there is an animal X2 then X1 loves the animal X2.
3. If there is somebody X1 and if there is an animal X2 then X1 loves the animal X2 then it is false that X1 kills an animal.
4a. Jack kills Tuna or Curiosity kills Tuna.
4b. Tuna is a cat.
5. It is false that Curiosity kills Tuna.
6. Every cat is an animal.

and the result when adding the negation of the query and submit it to *gkc* will be

```
result: proof found
for curiosity.fof
```

followed by the proof of unsatisfiability in 21 inference steps.

The second way to obtain the same result is to use Attempto's reasoner RACE[43] (Fig. 16.12). Just type the six sentences (without the negation of the query) in the "Axioms" field, go to the tab "Answer Query," and submit the query

[40] https://github.com/tammet/gkc#Compiling-and-installing

[41] https://www.tptp.org/

[42] http://attempto.ifi.uzh.ch/race/

[43] http://attempto.ifi.uzh.ch/race/

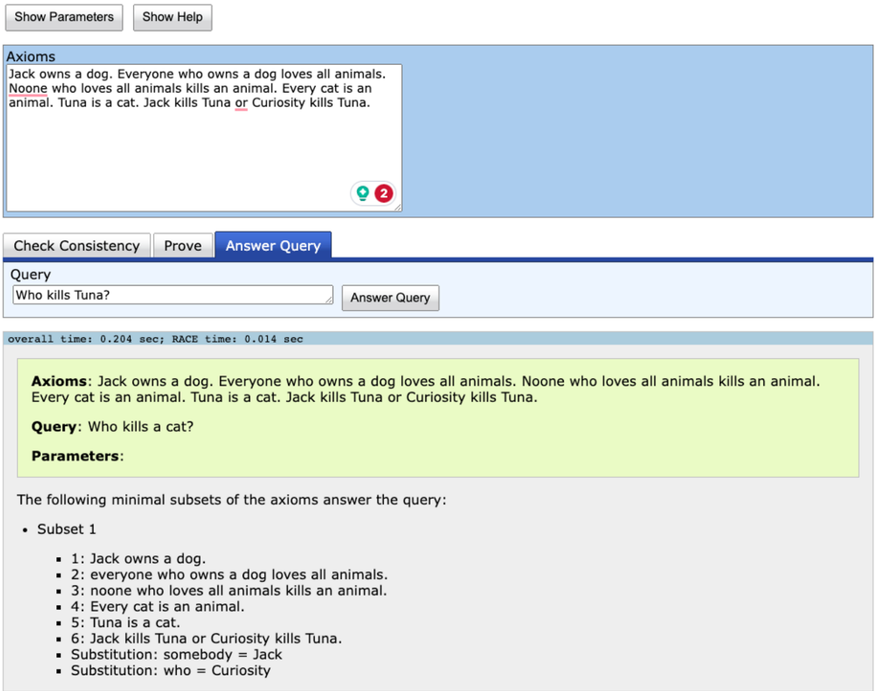


Fig. 16.12 Submitting the query "Who kills Tuna?" to RACE

Who kills Tuna?

The answer will be:

```
Substitution: who = Curiosity
```

where the `who` variable corresponds to the interrogative pronoun of the query.

Hints for Exercise 9-4: The "logical proof of the existence of God" fallacy

Think of the definition of the implication in terms of negation and disjunction (p. 277). Is it compatible with the English statement?

Read [31, Chapter 3][44].

[44] The book can be freely downloaded from https://link.springer.com/chapter/10.1007/978-1-4020-2712-3_3.

16.10 Chapter 10: Ontologies and Conceptual Graphs

Hints for Exercise 10-1: Translate DL formulas into Attempto and find counterexamples

On using Attempto Controlled English, see Exercises 6-1 and 9-3. *owl-cli* is a parser and diagram creator, binaries can be downloaded for each platform.[45]

Beware of the various OWL formats: Attempto outputs data in an OWL/XML format that is not recognized by *owl-cli*, but also in the Functional-Style syntax, which is recognized by *owl-cli* for drawing diagrams but not for making inferences. The inference engine of *owl-cli* expects data in the Turtle notation. Fortunately, an online conversion service is provided by the *Linked Data Finland* initiative at Aalto University.[46] Having our data make a roundtrip to Finland allows us to bridge the output of Attempto with the input of the *owl-cli* engine. ☞ 319

Inspecting Attempto's paraphrases and *owl-cli*'s diagrams is important to ensure the formula has been correctly translated into OWL. The visual syntax of the diagrams is described on *owl-cli*'s Web site.[47]

Expected Results

The first formula is $\exists \texttt{manages.Project} \sqsubset \texttt{Manager}$, which can be translated as

> Those who manage at least one project are managers.

> In Attempto Controlled English, this would be

> Every employee that manages a project is a manager. Jack is an employee. Jack manages a project. Jack is not a manager.

The reader can see the diagram produced by *owl-cli* in Fig. 16.13. Here are some explanations

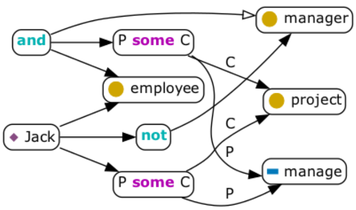


Fig. 16.13 The *owl-cli* diagram of "Every employee that manages a project is a manager. Jack is an employee. Jack manages a project. Jack is not a manager."

about the diagram: the green "and" on the top left is an intersection between those who manage some projects and employees. The white-headed arrow leaving the "and" is a subclass relation directed to managers. On the lower part of the diagram, we find the part that refers to Jack:

[45] https://atextor.de/owl-cli/main/snapshot/index.html

[46] https://www.ldf.fi/service/owl-converter/

[47] https://atextor.de/owl-cli/main/snapshot/diagram-notation.html

the three arrows leaving Jack are, from top to bottom, the fact that he is an employee, the fact that he is not a manager, and the fact that he belongs to those who manage some project. Once converted into TTL syntax, when we then run *owl-cli* with the `infer` command line option on the file containing the TTL syntax, we get the result

```
Error: Cannot do reasoning with inconsistent ontologies!
Reason for inconsistency: An individual belongs to a type and its complement
```

The reader has certainly recognized this individual: it's Jack. We have proved that Jack is a counterexample to $\exists\mathsf{manages.Project} \sqsubset \mathsf{Manager}$.

The other examples are treated similarly.

Hints for Exercise 10-2: Write a Q&A system for the Wildlife Ontology

The question patterns can be analyzed with *stanza*. Use *owlready2* to access the ontology through Python and *rdflib* to send SPARQL queries. URI issues can be pretty tricky; therefore do not hesitate to look inside the OWL code to find what is going on.

Expected Results

Here are the translations of the ontology content description sentences into Attempto Controlled English:

> Every tree is a plant. Every branch is a part of a tree. Every carnivore is an animal that eats some animals. Every leave is a part of a branch. Every giraffe is a herbivore. Every giraffe is an herbivore that eats nothing but leaves. Every herbivore is an animal that eats nothing but parts of some plants. Every lion is a carnivore that eats nothing but herbivores. No lion is a giraffe. No plant is an animal. Léo is a lion. Gigi is a giraffe. Giginou is a giraffe. Giginou is not Gigi. Léo eats Gigi.

When we run *stanza* on the three questions, and looking at the question variants, we conclude that we will be looking for the following conditions:

1. $\text{class}_{\text{NOUN}} \xrightarrow{\text{nsubj}} \text{“who”}_{\text{PRON}} \xleftarrow{\text{cop}} \text{“be”}_{\text{AUX}}$;
2. $\text{individual}_{\text{PROPN}} \xrightarrow{\text{nsubj}} \text{class}_{\text{NOUN}} \xleftarrow{\text{cop}} \text{“be”}_{\text{AUX}}$;
3. $\text{“about”}_{\text{ADP}} \xrightarrow{\text{case}} \text{individual}_{\text{PROPN}} \xrightarrow{\text{obl}} \text{“know”}_{\text{VERB}}$;

When we program this in Python we get the following answers (we have boldened the words that allowed us to detect the question pattern):

> Q: **Who are** the **giraffes**?
> A: There are 2 giraffes in AWO: Giginou, Gigi.
> Q: **Is Léo** a **lion**?
> A: Yes.
> Q: What do we **know about Gigi**?
> A: Gigi is a giraffe, which is a herbivore, which is an animal.
> Q: Tell me **who are** the bloody **lions**?
> A: There is one lion in AWO, namely Léo.
> Q: **Is Gigi** actually a **giraffe**?
> A: Yes.
> Q: What is there to **know about Léo**?

A: Léo is a lion, which is a carnivore, which is an animal.
Q: **Who are** the **whales**?
A: There is no class of whales in AWO.
Q: What do we **know about Davros**?
A: There is no Davros in AWO.
Q: Sorry for bothering you, but **is Giginou** by any chance a **Dalek**?
A: There are no Daleks in AWO.

16.11 Chapter 11: Unicode

Hints for Exercise 11-1: Steganography Using Unicode Combining Characters

Vietnamese uses ASCII vowels “a,” “e”, “i,“ “o,” “y,” and “u” as well as “ă” (“a” with breve U+0306), “â” (“a” with circumflex U+0302), “ê” (“e” with circumflex U+0302), “ô” (“o” with circumflex U+0302), “ơ” (“o” with horn U+031B) and “ư” (“u” with horn U+031B). These are the twelve base vowels. Furthermore, tones are indicated by diacritics: grave accent (U+0300, tone 2), acute accent (U+0301, tone 3), dot below the letter (U+0323, tone 4), question mark accent (U+0309, tone 5), tilde accent (U+0303, tone 6), and absence of diacritic (tone 1).

Unicode combining characters are encoded from the center outwards. For example, “ẫ” is encoded as an “a” base letter, followed by a circumflex combining accent and a tilde combining accent. Encoding the combining accents in the opposite order would result in the tilde being below the circumflex accent. When diacritics belong to different “stacks,” the order can be arbitrary. For example, “ậ” can be encoded arbitrarily as a base letter “a” followed by a circumflex accent, followed by a sublinear dot, or as a base letter “a” followed by a sublinear dot, followed by a circumflex accent.

A possible solution would be to write a Python list `COMPOSED` of all accented vowels (as precomposed Unicode characters) and a second Python list `DECOMPOSED` containing all possible “expansions” of the members of the previous list. For example, “ậ” would be a member of `COMPOSED`, and the corresponding member (same position in the list) of `DECOMPOSED` would be

```
["â\u0323","a\u0302\u0323","ạ\u0302","a\u0323\u0302"]
```

Do not forget to normalize the channel file (NFC) before hiding the payload. This can be ☞ 347
done using the *unicodedata* Python package.

If the capacity of the channel exceeds the length of the payload (plus two units used for the start and end of message marks), it is advised to introduce random pseudo-payload characters before and after the payload. These random characters should not include the “Message Start” character (value 27).

Expected Results

The result is that the Vietnamese UDHR can hold a payload of 467 characters. Out of these, we need two characters for the “Message Start” and “Message End” marks so that we have an effective payload of 465 characters. The length of the NFC-normalized file is 11,216 characters, so we have a density of 465/11,216 = 0.04146. In other words, we need, on average, 24.12 Vietnamese graphemes to encode a single payload character.

If we convert *Moby Dick* into "telegraphic" format (uppercase letters and blank spaces), the first 465 characters correspond to the following words:

> CHAPTER LOOMINGS CALL ME ISHMAEL SOME YEARS AGO NEVER MIND HOW LONG PRECISELY HAVING LITTLE OR NO MONEY IN MY PURSE AND NOTHING PARTICULAR TO INTEREST ME ON SHORE I THOUGHT I WOULD SAIL ABOUT A LITTLE AND SEE THE WATERY PART OF THE WORLD IT IS A WAY I HAVE OF DRIVING OFF THE SPLEEN AND REGULATING THE CIRCULATION WHENEVER I FIND MYSELF GROWING GRIM ABOUT THE MOUTH WHENEVER IT IS A DAMP DRIZZLY NOVEMBER IN MY SOUL WHENEVER I FIND MYSELF INVOLUNTARILY PAUSING

which amounts to seven and a half lines of the 1851 edition.[48] As pages in this edition contain 34 lines of text, this extract corresponds to 22% of a page.

The entire (telegraphic) *Moby Dick* text consists of 1,150,306 characters. For a payload of this size, one would need a Vietnamese text of approx. 28 MB.

Let us investigate the effect of compression. When we compress the (standard) *Moby Dick* text in 7z, we get a file of 400,783 bytes (a compression rate of 66.37%). As the 7z file uses bytes, we need 8 bits per character. We said that we could fit 467 5-bit characters into the Vietnamese UDHR. This means that we have $467 \times 5 = 2{,}335$ available bits, so that we can encode $\frac{2{,}335}{8} = 292$ 8-bit characters. In other words, we need, on average, 38.4 Vietnamese characters to encode a single 8-bit character. Therefore for the entire 7z-compressed *Moby Dick*, we would need only 15.39 MB of Vietnamese text, which is quite an improvement.

We have downloaded Vietnamese Wikipedia[49], a 6.09 GB XML file, and applied 🗋`viet-capacity.py` to it. The result (obtained after 2h41′ on a MacBook Pro 2,3 GHz Intel Core i9 8 cores with 32 MB of RAM) is that we have a payload of 61,885,471 5-bit characters, i.e., approx. 309.4 million bits. This means we can encode 53 times the simplified (5-bit) *Moby Dick*, 32 times the uncompressed standard *Moby Dick*, and 96.5 (!) times the 7z-compressed standard *Moby Dick*.

Let us now answer questions 4 and 5. The density could be improved if we refine our carrier. We have seen that the precomposed character "ậ" can have four different decomposed versions. Suppose we attach a different value to each, plus the precomposed version. In that case, we get five values for a single Vietnamese grapheme instead of the two values we used (corresponding to "entirely precomposed" and "not entirely precomposed").

As for question 5, Akbas [1] proposes the use of ZWJ and ZWNJ characters, which are mostly invisible when used in English texts (careful readers may observe broken ligatures in some cases when ZWNJ characters are used). This method can indeed have a very high density: after all, one can use eight ZWJ or ZWNJ characters for each "visible" English grapheme, resulting in a density of 100%, that is, having as many hidden characters as visible ones. In comparison, our method has a density of only 4.16%.

But there is a vital hitch: ZWJ and ZWNJ characters are not supposed to be used in English text in the first place. Their presence is an eye-popping hint to the fact that something is wrong. In other words, Akbas's method may be efficient in storing a lot of information but breaks the undetectability condition.

On the other hand, our method is much more difficult to detect. Let us compare the statistics of the original Vietnamese UDHR, as retrieved from `https://unicode.org/udhr` with the statistics of our channel:

[48] `https://archive.org/details/MobyDick1851HermanMelville`

[49] From `dumps.wikipedia.org` (search for `viwiki`).

	Original			Steganographic		
Nb. of variants	1	2	4	1	2	4
Precomposed	420	0	0	665	348	184
Decomposed	864	687	368	619	339	184

In this table, by the number of variants, we mean the number of different decompositions of the same precomposed character. As we see, in the original file, there are no precomposed “complex” characters (those having two or more decompositions), but “simple” characters (with only one possible decomposition) are 33% precomposed and 67% decomposed. In our case, there is a balance between precomposed and decomposed characters. The difference between the two is crisp and clear. Still, if we consider character composition to be a random feature of characters, then the original file is an exception, and ours is a perfectly natural example of Vietnamese Unicode-encoded text.

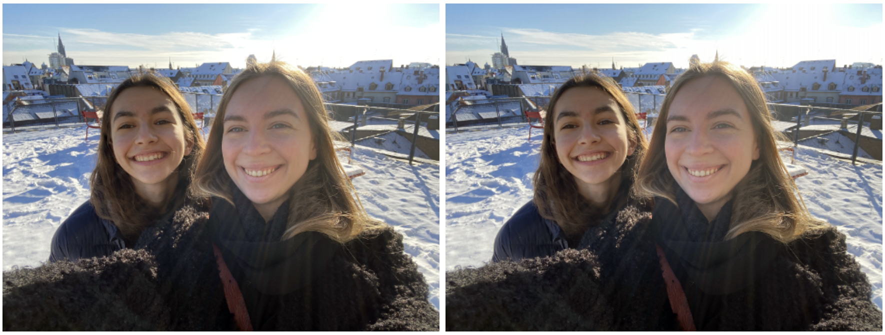


Fig. 16.14 A picture of Danaé (on the left) and Ernestine (on the right). The left one is the original, and the right one is such that color values in odd-numbered pixels are odd, and color values in even-numbered pixels are even.

To conclude this section, let us highlight the fact that text steganography may seem impressive but is actually peanuts compared to what can be hidden in images. In Fig. 16.14, we display two images of $4,032 \times 3,024$ pixels each, taken by an iPhone 7 Plus. The left is the original (14,431,658 bits), and in the right one (15,530,007 bits), we have replaced the three color values of each odd-numbered pixel with the closest odd values, and the three color values of each even-numbered pixel with the closest even values. As the reader can see, the two images are *visually indistinguishable*. The image on the right is our channel: let $\phi(3j+i)$ be the value of the i-th color (for $i \in \{0, 1, 2\}$, meaning the red, green, and blue) of pixel j; if j is odd then $\phi(3j+i)$ must be odd as well. To insert information, we change this to the closest even value when we want to insert a 1, and we leave it as is when we want to insert a 0. This image contains 36,578,304 color values, which allows us to encode 4,572,288 8-bit characters (plus a few characters to indicate the beginning and end of the data), that is 3,8 times the entire *King James Bible* in 7z compression. And it is not even a large image: the standard image size of the current (as of 2023) Apple iPhone devices is $8{,}064 \times 6{,}048$, which is four times the size of our sample image.

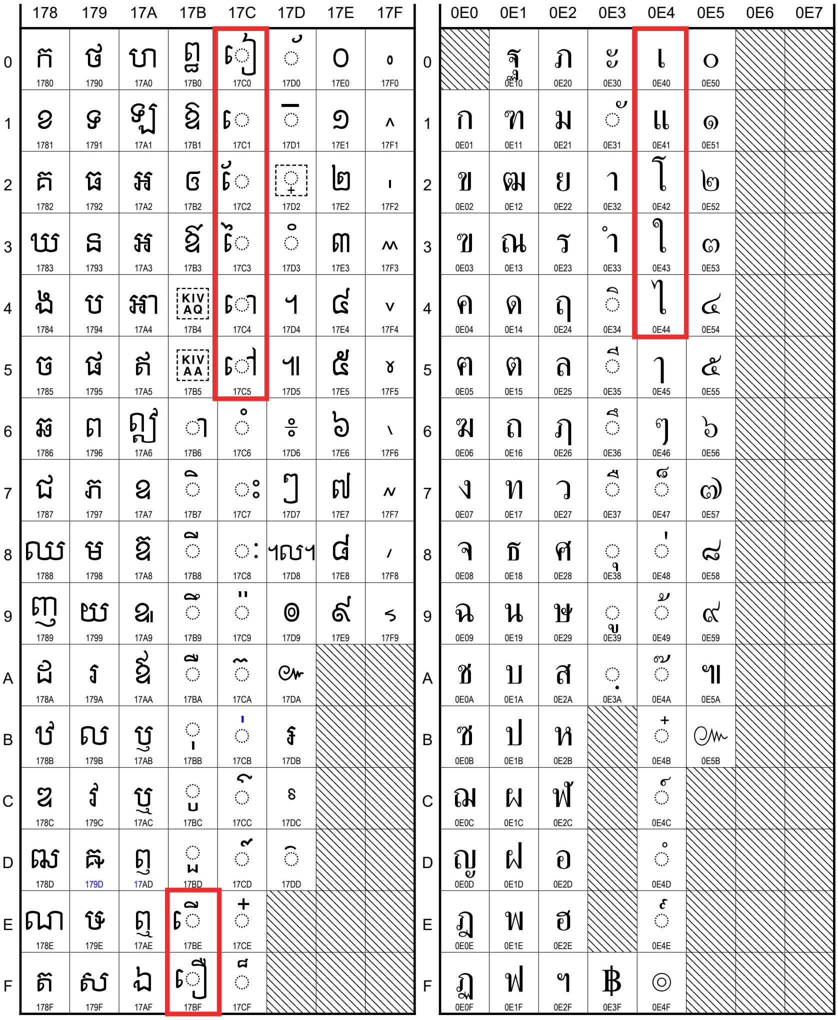


Fig. 16.15 Unicode charts of Khmer (on the left) and Thai (on the right). © 1991-2023 Unicode Inc. Reproduced with the permission of the Unicode Consortium (Unicode, Inc.). For Terms of Use and further information, see `https://unicode.org/charts/PDF/U1780.pdf`.

Hints for Exercise 11-2: Southeast-Asian Scripts Implementations in Unicode

Look at positions U+17BE–17C5 for Khmer and U+0E40–0E44 for Thai.

Expected Results

Fig. 16.15 displays the Unicode charts of Khmer and Thai. The symbol "◌" means that a ☞ 347 character is *combining*, i.e., that its glyph combines with the glyph of the previous character.

From a linguistic point of view, what graphemes U+17BE–17C5 in Khmer and U+0E40–0E44 in Thai have in common is that they are dependent vowels (vowels that can only be used in combination with a consonant) that *are placed on the left of the consonant even though they are pronounced after the consonant*. According to Unicode's 6th Design Principle,

> The default for memory representation is logical order. [2, p. 14]

Of course, the term "logical order" does not make sense in linguistics, where logic applies only to the higher strata of language, such as semantics. What is actually meant (see also [18]) is

> The order in which Unicode text is stored in the memory representation is called logical order. This order roughly corresponds to the order in which text is typed in via the keyboard and to phonetic order. [2, p. 19]

In other words, the logical order is the order of phonemes mapped to graphemes. Eight Southeast-Asian scripts do not obey the logical order principle: Thai (U+0E00–0E7F), Lao (U+0E80–UEFF), Tai Le (U+1950–197F), New Tai Lue (U+1980–19DF), Tai Viet (U+AA80–AADF), Nyiakeng Puachue Hmong (U+1E100–1E14F), Pau Cin Hau (U+11AC0–11AFF), Hanifi Rohingya (U+10D00–10D3F). To verify this property look at dependent vowels "E" in Unicode: U+0E40, U+03C0, U+196B, U+19B5, U+AAB5, U+1E12A, U+11AD6, U+10D20.

On the contrary, scripts such as Javanese (U+A980–A9DF), Balinese (U+1B00–1B7F), and Indian scripts such as Devanagari (U+0900-097F), and Sinhala (U+0DC3–0DF4), use the logical order. All scripts of that area derive from the Brahmi script (U+11000–1107F), in which the dependent vowel U+11042 (the ancestor of the graphemes mentioned) is still simply a diacritic, but it is clearly placed on the left part of the consonant. Its shape later evolved to become a full-size grapheme on the left of the consonant.

As for the frequency of these vowels, they amount to 7.47% of the total number of graphemes in Thai and 6.14% in Khmer (as counted in the Thai and Khmer versions of the Universal Declaration of Human Rights[50]).

The reason for this discrepancy between practices is that in the case of Thai, there was already a Thai national standard (TIS-620[51] aka ISO 8859-11). TIS-620 was released in 1990, one year before the publication of the first version of the Unicode standard, so, according to Unicode's 10th Design Principle, which is "Convertibility," switching to logical order would hinder convertibility between TIS-620-encoded and Unicode-encoded texts.

As [15, p. 612] points out, even though Thai does not use logical order in the encoding of text, it uses logical order for collation: to sort Thai text, one must first invert consonant and ☞ 348
dependent vowel, and *then* sort.

Hints for Exercise 11-3: How do Unicode and Wikipedia deal with simplified and traditional Chinese?

Read [16], [2, pp. 746–747], as well as the "Automatic conversion between simplified and traditional Chinese" page on Wikimedia.[52]

---

[50] https://unicode.org/udhr/translations.html

[51] https://itscj.ipsj.or.jp/ir/166.pdf

[52] https://meta.wikimedia.org/wiki/Automatic_conversion_between_simplified_and_traditional_Chinese

Expected Results

In 1990, the original ISO 10646 encoding, an encoding of 128 groups of 256 planes of 256 rows of 256 cells, proposed different planes for simplified Chinese, traditional Chinese, and Japanese kanji. Unicode, instead, unified sinograms into a single table and therefore claimed to encode all scripts of the world in only one plane of 65,536 characters—a claim it withdrew a few years later when Unicode was extended to 17 planes (but 17 is still better than 32,768). The industry preferred Unicode, and ISO 10646-1 (and all subsequent versions of it) became a carbon copy of Unicode.

The unification of sinograms by Unicode was based on three rules [2, pp. 749–752]:

1. Source Separation Rule. If two ideographs are distinct in a primary source standard, then they are not unified.
2. Noncognate Rule. In general, if two ideographs are unrelated in historical derivation (noncognate characters), then they are not unified.
3. By means of a two-level classification ([2, pp. 749–752]), the abstract shape of each ideograph is determined. Any two ideographs that possess the same abstract shape are then unified, provided that their unification is not disallowed by either the Source Separation Rule or the Noncognate Rule.

As for the simplified/traditional distinction, the two characters are obviously cognates but mostly do not have the same abstract shape (otherwise, there would be no simplification), and most importantly, they belong to different encoding standards.

So the answer to the first question is: yes, Unicode distinguishes between traditional and simplified characters, but they do not belong to different tables: all sinograms (called CJKV characters by Unicode) are stored together.

Nevertheless, this doesn't mean that Unicode covers entirely the simplification reforms. Indeed, the reforms contained a certain number of simplified components, meaning that every character that contains one can be simplified. As stated in [2, p. 747]:

> The Unicode Standard does not explicitly encode all simplified forms for traditional Chinese characters. Each should be used appropriately where the simplified and traditional forms exist as different encoded characters. The Unicode Standard does not specify how to represent a new simplified form (or, more rarely, a new traditional form) that can be derived algorithmically from an encoded traditional form (simplified form).

As for Wikipedia, the short answer is: no, there is only one Chinese Wikipedia. It is an astonishing accomplishment by Chinese-speaking Wikipedians to have built a single Wikipedia [21], and not a small one: as of October 2023, it features 1.38 million articles written by 3.41 million people, and it ranks 13th worldwide in terms of number of articles.

Chinese Wikipedia uses an automatic conversion algorithm written in PHP[53]. As explained in a Chinese Wikipedia help page,[54] there are several layers of word conversion: a built-in conversion table[55], a collection of "public conversion groups," meaning Wikipedia pages with

[53] https://phabricator.wikimedia.org/source/mediawiki/browse/master/includes/language/converters/ZhConverter.php

[54] "Help:中文维基百科的繁简、地区词处理"ZH ⌊Help:Chinese Wikipedia's handling of traditional, simplified, and regional words⌋ https://zh.wikipedia.org/wiki/Help:%E4%B8%AD%E6%96%87%E7%BB%B4%E5%9F%BA%E7%99%BE%E7%A7%91%E7%9A%84%E7%B9%81%E7%AE%80%E3%80%81%E5%9C%B0%E5%8C%BA%E8%AF%8D%E5%A4%84%E7%90%86. An automatically translated version of the document is available here: zh-wikipedia.pdf.

[55] https://phabricator.wikimedia.org/source/mediawiki/browse/master/includes/language/converters/ZhConversion.php

translations of terms or toponyms, and "manual conversions," that is the possibility of giving the precise conversion intended by the author. It is, therefore, possible to deactivate conversion or to give specific conversions. For example, to keep text as it is, one uses `-{R|㐷}-`, to specify specific versions, one can use `-{T|zh-hans:㐹;zh-hant:㑶;}-` where `zh-hans` refers to simplified Chinese and `zh-hant` to traditional Chinese. There is also the instruction `__NOCC__` to deactivate autoconversion entirely.

The conversion table looks like this:

```
class ZhConversion {
        public static $zh2Hant = [
                '㐷' => '傌',
                '㐹' => '㑶',
...9,570 similar lines...
                ',并力讨' => ',并力討',
                ',个中' => ',箇中',
        ];

        public static $zh2Hans = [
                '㑯' => '㑔',
                '㑳' => '㑇',
...4,643 similar lines...
                '蕒' => '荬',
                ',陞' => ',升',
        ];
}
```

where simplified is on the left and traditional on the right in the first table, and the other way around in the second table. From simplified to traditional is the most difficult direction since many simplified characters correspond to more than one traditional character [16]. This explains the difference in the size of the two resources. We compared the results of *opencc* and *hanziconv* with those of the Wikimedia algorithm. Here are the results:

	Simp → Trad			Trad → Simp		
	wrong	correct	accuracy	wrong	correct	accuracy
OpenCC	1,900	7,674	80.15%	2,870	11,351	79.82%
HanziConv	4,518	5,056	52.81%	6,450	7,771	54.64%

Hints for Exercise 11-4: Unicode and political correctness in the processing of emoji

Read [18, pp. 155–159] and some newspaper articles.[56,57]

Expected Results

There are three axes of political correction that Unicode has considered when defining emoji: gender, the color of skin, the color of hair. In all cases, Unicode has defined a "non-realistic"

[56] https://perma.cc/2T8Z-JD22

[57] https://perma.cc/2QVY-6994

version that can be used as a neutral feature and multiple realistic features. The non-realistic version of "man" or "woman" is called "adult." Skin color and hair color are indicated by adding special variation characters. When no variation character is used, skin and hair color is yellow (admittedly, non-realistic).

Fig. 16.16 Apple Color Emoji versions of emoji x1F468 MAN, 0x1F9D1 ADULT, and 0x1F469 WOMAN

Unicode's intentions are undoubtedly pure, but the risk of lack of political correction is still present since font designers are free to represent neutral or non-neutral emoji in their own way. In particular, there is a risk of Western cultural bias. The reader can see in Fig. 16.16 the three emoji, 0x1F468 MAN, 0x1F9D1 ADULT, and 0x1F469 WOMAN, as represented in the *Apple Color Emoji* font. The "woman" representation has blond hair and dyed lips, while the "man" representation has short hair and carries a mustache—these aspects seem suspiciously Western-like. Fortunately, fonts change more easily than encodings. We'll see what the future holds for political correctness.

Hints for Exercise 11-5: Curiosa Unicodensis

1. See [2, p. 883];
2. see [2, p. 773];
3. see [2, p. 937] or [15, pp. 134–136];
4. see [2, p. 287] or [15, p. 349];
5. see [2, p. 293].

Expected Results

1. There are two kinds of invisible characters in Unicode: those that affect the rendering of ☞ 347 glyphs (like ZWJ and ZWNJ) and those that carry specific semantics targeted to some other application than text renderers. This is the case of the three invisible characters: their *raison d'être* is to disambiguate mathematical formulas: when we write $f(x)$, it is a worldwide convention to assume that f is a function, x is a variable, and f is applied to x. Humans are sufficiently aware of the context to implicitly verify whether this assumption is true every time they see $f(x)$. For a machine, it is better to use U+2061 FUNCTION APPLICATION between f and (. When writing ab instead of $a \times b$, a U+2062 INVISIBLE TIMES removes the ambiguity. Similarly, when writing a_{ij} instead of $a_{i,j}$, one will use U+2063 INVISIBLE SEPARATOR;
2. ｆｕｌｌ－ｗｉｄｔｈ ｃｈａｒａｃｔｅｒｓ ｌｉｋｅ ｔｈｅｓｅ exist because sinographic writing systems are very keen on aligning s, especially when writing verti-

cally. Alignment is guaranteed for sinographic graphemes, but what happens when some words in the Latin alphabet appear among the sinograms? This is why full-width s are used, and Unicode contains a complete copy of the ASCII table for this purpose;

3. *surrogates* (high and low) were a convenient solution in the early days of Unicode when it was encoded in 16 bits. By reserving 1,024 positions for being initial sequences (“high” surrogates) and another 1,024 positions as being final sequences (“low” surrogates), and by combining them, one could access positions outside the 16-bit range. When Unicode was extended to 17 planes, and UTF-8 was adopted, surrogates lost their appeal and are now not used anymore;
4. the GETA MARK 〓, from the Japanese “下駄”$_{\text{JA}}$ ⌊wooden sandal⌋ was invented by Japanese printers as a means to replace missing τ-graphs, at least until they could be replaced. When τ-graphs were missing in the printing process, the printer would order them from a foundry. During the time these τ-graphs were prepared, she would use a geta mark in the proofs of the text: it is eminently visible and takes the same space as any other sinographic τ-graph;
5. when a character has the “soft dotted” property, it means that it has a diacritic that is “soft” in the sense that if you place another diacritic on it, it will disappear. This is the case of the letter ‘i,’ which becomes, e.g., ‘î’ when we endow it with a circumflex accent. In some cases, we need a “hard-dotted i.” To obtain this effect, Unicode suggests taking a regular (soft dotted) ‘i’ and applying a dot combining character to it. The “hard” dot will then smash the “soft” dot, and we end up with a ‘hard-dotted i.’ The case of Turkish is different: it has a grapheme represented by an undotted ‘ı’ and another grapheme represented by the dotted ‘i.’ The γ-graph of the Turkish ‘dotted i’ grapheme has the soft dotted property since there are words like “dâhilî”$_{\text{TK}}$ ⌊internal⌋, where the circumflex accent has smashed the dot. There is no ambiguity with ‘ı,’ which is never accented.

16.12 Chapter 12: XML, TEI, CDL

Hints for Exercise 12-1: Jerome K. Jerome’s *Three Men on the Bummel* in TEI

Complete the TEI header using information from the original edition of the book[58]. Notice that it has two publishers (at different locations), belongs to a series, and has an author and an illustrator. Notice that the text fragment belongs to Chapter XIII of the book and that at the beginning of the chapter, we have placed a reference target (carrying the ID `Korps-chap-13`) concerning the definition of a Korps (a fraternity of students in early 20th century Germany). The text contains 16 segments in a foreign language. There is a mistake in one of them, which could be marked by a `sic` element: “stoffe” should be “Stoffe,” since “Stoff”$_{\text{DE}}$ ⌊fabric⌋ is a noun.

[58] https://archive.org/details/threemenonbummel00jerouoft/page/296/mode/2up

Hints for Exercise 12-2: Find the subject occurrences in Bach's Fugue BWV 846

The XML data of our fugue is contained in file BWV_0846_fuga.xml, which we have adapted from data available on the *Tobis Notenarchiv*[59]. In the MusicXML format, a note is described as follows:

```
<note id="n213">
    <pitch>
        <step>B</step>
        <alter>-1</alter>
        <octave>3</octave>
    </pitch>
    <duration>120</duration>
    <voice>4</voice>
    <type>16th</type>
    <accidental>flat</accidental>
    <stem>down</stem>
    <staff>2</staff>
</note>
```

which is very simple: this note is identified as n213, its pitch is a B♭3 (a B-flat in the third octave), its duration is 120 units (the complete measure being 1,920, therefore it is a $\frac{1}{16}$th of the measure, i.e., a sixteenth, called "semiquaver" in the UK). It belongs to the 4th voice of the fugue (the lowest, also called "bass," as is usual with singers: soprano, alto, tenor, bass). The rest is purely graphic: the stem is underneath the note and written on the bottom staff (as it is usually played by the left hand on a keyboard instrument).

The subject is a sequence of notes repeated many times in the fugue by all voices. But there is a snag to it: the subject is played in different keys, so that when it first appears (by the second voice), it is C-D-E-F-... but when taken over by the first voice, it is G-A-B-C-…, and later on, it becomes E-F♯-G♯-A-…, and also A-B-C♯-D-…, and even F-G-A-B♭-…, etc.

Find a way of rewriting the sequence of notes of every voice so that a regular expression can be applied to detect the subject, *no matter in what key it is written*.

Use XML DOM to extract information from the XML file and SAX to inject information about color so that in the XML file, every detected occurrence of the subject is colored according to the voice to which it belongs (using red, green, blue, and magenta for the four voices). To visualize the score and export it in PDF form, you can use *MuseScore*,[60], a fantastic piece of free software.

Expected Results

Our first challenge is to rewrite the sequence of notes of each voice in such a way that the subject corresponds to the same sequence of symbols so that a regular expression can detect it, no matter in what key it is written.

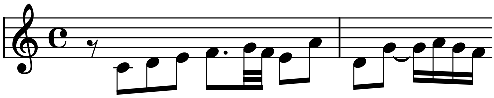
Let's think for a minute. What do all the occurrences of the subject have in common? It is the relation between the individual notes: we start at some pitch, and then we go up for a tone, and then another tone, and then a semitone, and so on. This is independent of the key, and so

[59] https://tobis-notenarchiv.de/wp/en/bach-archive/instrumental-works/keyboard-works/the-well-tempered-clavier-book-i/

[60] https://musescore.org/

is the duration of each note. What we need is a *relative* notation, which gives a starting pitch (at the beginning of each voice) and then only gives relative raises and falls in semitones (on which most Western music is based). Such a relative notation is used in Byzantine music [30].

We will, therefore, start each voice with an absolute pitch (for the second voice, it happens to be C_4). Then we will write a $\nearrow$ symbol for every semitone of pitch raise, a $\searrow$ symbol for every falling semitone, and an equal sign whenever the same note is played again. We will also add symbols for the duration, using multiples of 60 units: ^{1} for a 32nd, ^{2} for a 16th, …, ^{16} for a half-measure and ^{32} for a full measure. The (first occurrence of the) subject, which is



is now written

$$R^4C_4^4 \nearrow\nearrow^4\nearrow\nearrow^4\nearrow^6\nearrow\nearrow^1\searrow\searrow^1\searrow^4\nearrow\nearrow\nearrow\nearrow\nearrow^4\searrow\searrow\searrow\searrow\searrow\searrow\searrow^4\nearrow\nearrow\nearrow\nearrow\nearrow^4{=}^2$$
$$\nearrow\nearrow^2\searrow\searrow^2\searrow\searrow^2\searrow^2$$

where R stands for “rest,” and C_4 is the first note of the voice.

There is another difficulty: imagine searching for the expression above (without the R^4 and C_4^4) and finding it at several locations in the character string representing the voice. How do we know to what note they correspond in the XML file? There is only one solution: to create an index file, mapping offsets in the string to notes in the XML file.

The reader can see the result in Fig. 16.17 and 16.17.

Discussion of the result

As can be seen in Fig. 16.17 and 16.17, our algorithm identifies 21 occurrences of the subject, some of them only partially.

According to [28] and Keller [19] (speakers of German will love reading David [11]), there are, in fact, 24 occurrences of the subject (as many as the number of the Preludes and Fugues in each volume of the *Well-Tempered Clavier*), so we have missed three of them. What happened?

The three missing subject occurrences are:

- second half of measure 17 and first half of measure 18, 4th voice, D_3-E_3-F_3-G_3 A_3-G_3-F_3 $B\flat_3$ E_3 A_3;
- second half of measure 19 and first half of measure 20, 2nd voice, E_4-$F\sharp_4$-G_4-A_4 B_4-A_4-G_4 C_5 $F\sharp_4$ B_4;
- second half of measure 21 and first half of measure 22, 2nd voice, B_3-C_4-D_4-E_4 F_4-E_4-D_4 G_4 C_4 F_4.

These are, in fact, modulations of the original subject since the pitch differences are not the same. In the original subject, we have 2, 2, 1, 2, −2, −1, 5, −6, 5 semitones. In the missing occurrences, we have instead: 2, **1**, **2**, 2, −2, −**2**, 5, −6, 5; 2, **1**, **2**, 2, −2, −**2**, **7**, −6, **7**; **1**, 2, **2**, **1**, −**1**, −**2**, 5, −**7**, 5 semitones (those that are different are typeset in bold). To identify them, we would need a more flexible algorithm, using not a precise match but a fuzzy match based on Levenshtein distance or some similar process.

Of course, as always in classification, if we adopt such a process, we will increase our recall, but precision may suffer since we will potentially identify also sequences that are not subject occurrences.

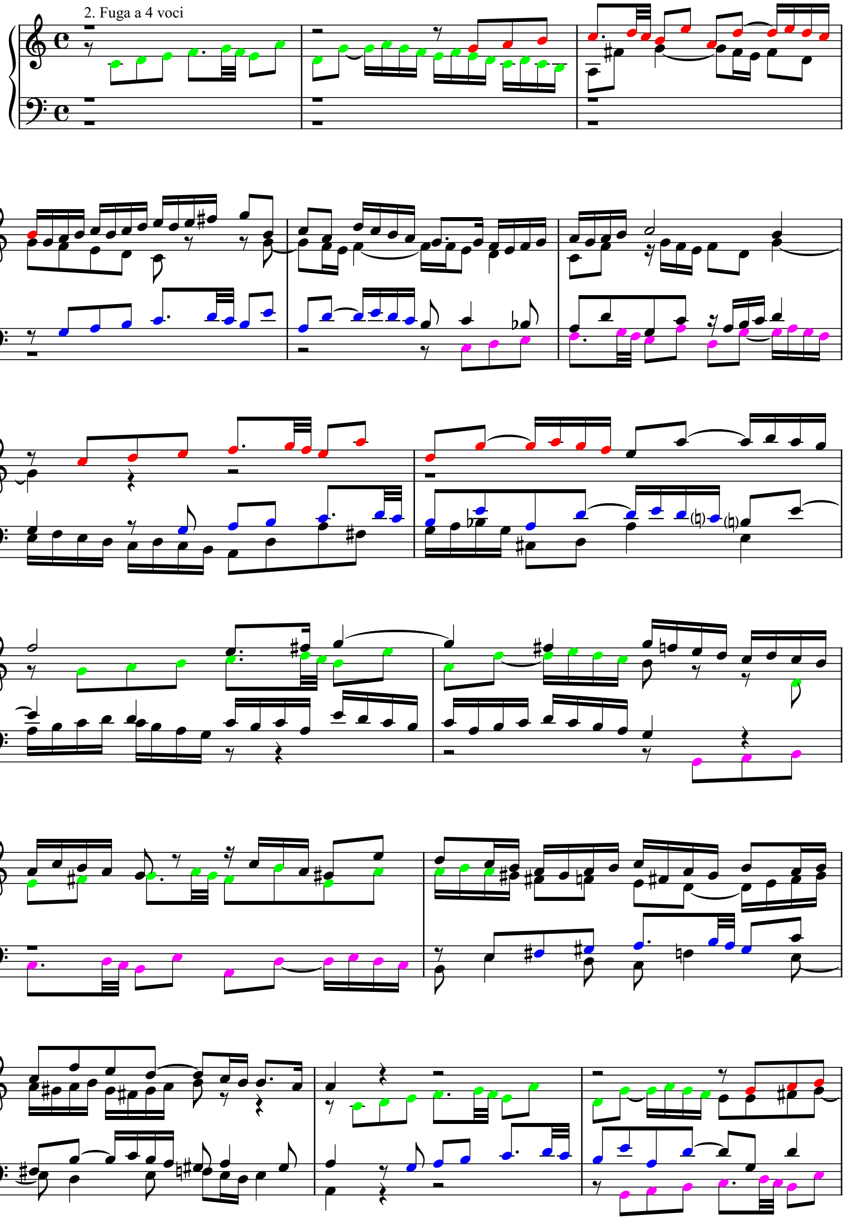



Fig. 16.17 The first page of the PDF output of the colored BWV 846 fugue, rendered by *MuseScore*

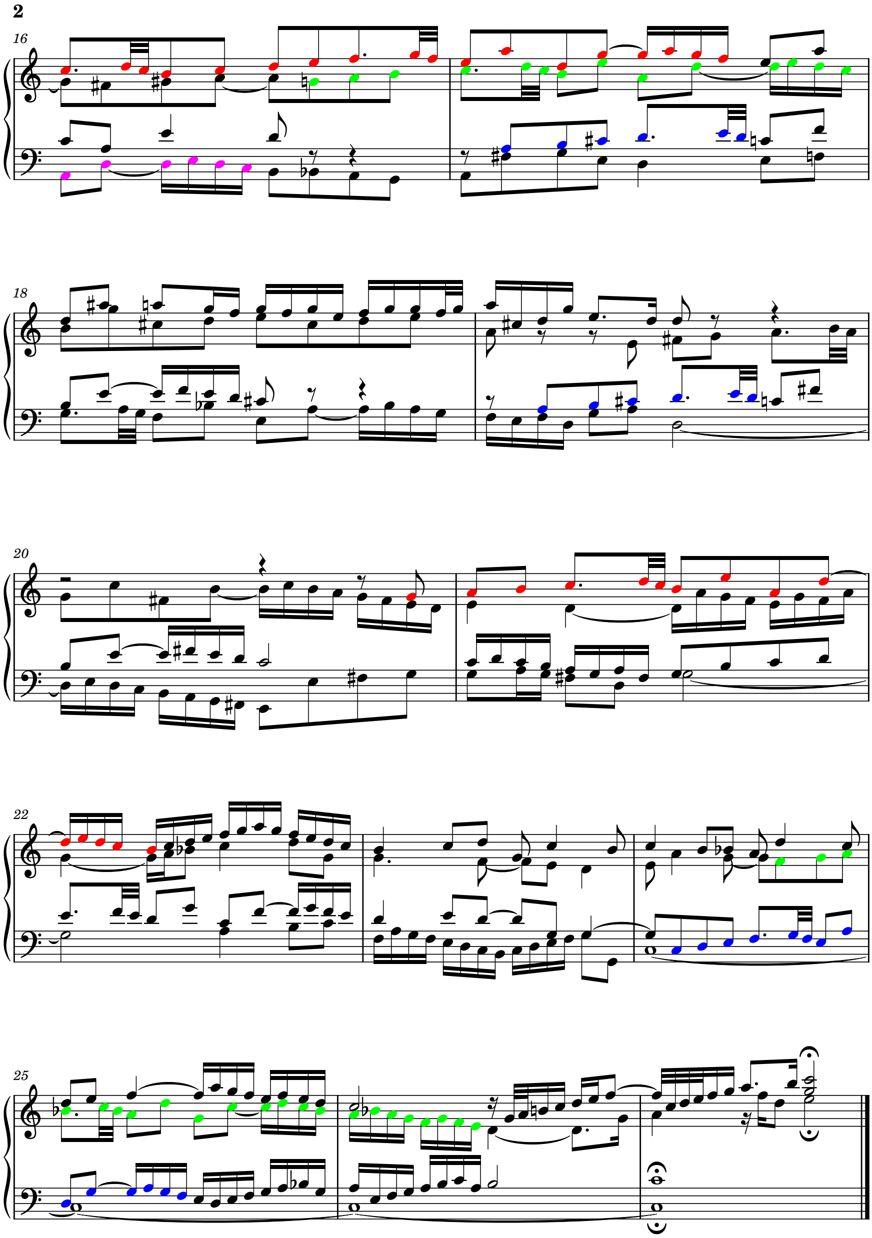


Fig. 16.18 The second page of the PDF output of the colored BWV 846 fugue, rendered by *MuseScore*

Homework: Listen to this fugue as played by your favorite interpreter and on your favorite instrument, and try to detect every occurrence of the subject.

Hints for Exercise 12-3: Document conversion from HTML to TEI using SAX and DOM

A minimal TEI document is of the form

```
<?xml version="1.0" encoding="utf-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title/>
    </titleStmt>
    <publicationStmt>
      <publisher/>
    </publicationStmt>
    <sourceDesc>
      <p/>
    </sourceDesc>
  </fileDesc>
</teiHeader>
<text>
<body>
   ...
</body>
</text>
</TEI>
```

Document subdivisions are represented by `div` elements inside `body`. A `div` element can carry a `type` attribute (with values `level1`, `level2`, etc.) and an `xml:id` attribute that takes unique XML names as values. The title of a subdivision is given by a `head` element, which must be the `div` element's first child.

HTML does not encode subdivisions but only subdivision titles. The challenge in the requested conversion is to find where a subdivision ends. We will follow the rule that an *n*-level subdivision closes all previous subdivisions of a level greater or equal to *n*.

To create a table of contents in TEI, use the following structure:

```
<front>
<div type="contents">
<head>Contents</head>
<list>
  <item><ref target="thraupidae">Thraupidae</ref>
    <list>
      <item><ref target="poospiza">Poospiza</ref>
  ...
</list>
</div>
</front>
```

The element `front` is placed under `text`, before `body`.

16.13 Chapter 13: Counting Words

Hints for Exercise 13-1: Exploring the Anthology of the Association for Computational Linguistics

From the Web site of the ACL Anthology,[61] the reader can download a BiBTEX file of all papers in two versions: with and without abstracts.

Use a Python package to extract author names from the BibTEX file without abstracts. Build a graph of co-authorships for each decade (from the fifties to the 2020s) and find the largest connected component and, inside this component, the most central author. Repeat this calculation for every decade (from the fifties to the 2020s). Calculate also the number of papers co-authored by each author, the number of co-authors, as well as the average number of co-authors per paper.

Read the BibTEX file with abstracts and extract the identifier, the year, and the abstract. Beware of papers written in languages other than English. Apply a keyword-extracting utility to abstracts, build a graph of the cooccurrence of keywords, find communities of keywords, and find the 20 most central keywords in each community.

Expected Results

The results are as follows (decade, number of authors, largest connected component, most central author, number of papers co-authored, number of co-authors, average number of authors per paper):

	# auths	larg. cc.	most central author	pap.	co-a.	avg.
1950s	29	22	Léon Dostert (Johns Hopkins University, Baltimore, MA)	4	21	8.50
1960s	136	22	Harry H. Josselson (Wayne University Detroit, MI)	5	14	4.00
1970s	169	8	Alan K. Melby (Brigham Young University, Provo, UT)	2	7	4.50
1980s	1,408	173	Louis des Tombe (Utrecht Institute of Linguistics)	8	15	4.88
1990s	4,333	2,083	Sergei Nirenburg (University of Maryland, Baltimore, MD)	41	47	3.46
2000s	13,547	10,336	Chu-Ren Huang (The Hong Kong Polytechnic University)	70	103	3.99
2010s	33,562	28,641	Pushpak Bhattacharyya (Indian Institute of Technology Bombay)	226	238	3.82
2020s	40,411	34,123	Pushpak Bhattacharyya (Indian Institute of Technology Bombay)	101	161	4.15

(The figures of the 2020s are partial since the data were retrieved on August 2023.) Many interesting facts can be read from this table. In the early years, the largest component was relatively small (in the 1970s, it was tiny, so the most central author has written only two

[61] https://aclanthology.org/

papers with seven co-authors). Still, starting with the nineties, it gets more and more critical, reaching 85% in the 2010s. Authors tend to become more and more prolific: Bhattacharyya wrote 226 papers in 10 years with 238 distinct co-authors, a performance that is comparable to the one of the famous mathematician Paul Erdős who wrote 1,500 papers with 500 co-authors, during 6.5 decades of scientific activity. Interestingly, the average number of co-authors remains invariant throughout seven decades and is a characteristic of the scientific domain.

We have tested two packages for keyword extraction from the abstracts: *summa* (the TextRank method) and *keybert*. The latter is based on BERT and would probably provide better results, *but* it required about 2 seconds for each abstract, which amounts to about 24 hours of calculations for the whole corpus. On the contrary, the former parsed all abstracts in only 4 minutes.

We keep only keywords that appear at least 100 times (line 43), build the graph of co-occurrences (line 44), and calculate communities using the weight of frequency of co-occurrence (line 46). The graph we obtain has 1,267 nodes and 105,125 edges, meaning it is quite dense. Six communities are detected, and it turns out that they are quite relevant. Here are the 20 most central keywords of each community (as well as the community's size between parentheses):

COMMUNITY 1 (370): languages, language, data, translation, annotated, annotation, corpus, annotations, different, english, nlp, annotators, word, resources, speech, linguistic, research, new, processing, texts;

COMMUNITY 2 (295): models, training, tasks, model, learning, performance, modeling, neural, domains, trained, attention, classification, representations, labels, language models, method, dialogue, pre, prediction, sentence;

COMMUNITY 3 (217): datasets, based, features, media, approaches, detection, approach, existing, analysis, study, studies, f, social, context, sentiment, experimental, framework, extensive, emotions, experiments;

COMMUNITY 4 (188): entity, entities, information, methods, knowledge, questions, answers, answering, relations, extraction, documents, graph, retrieval, dataset, summarization, domain, qa, question, supervised, summary;

COMMUNITY 5 (119): generation, evaluation, generated, generating, human, text, generate, evaluate, evaluating, generative, conversational, metrics, generates, generator, evaluated, images, conversations, evaluations, responses, visual;

COMMUNITY 6 (78): words, semantic, sentences, dependency, parsing, embeddings, similarity, syntactic, dependencies, structures, representation, structural, embedding, structure, parsers, unsupervised, vector, structured, meaning, space.

The first community deals with corpora processing, lexicons, WordNet, FrameNet, and the like; the second community is the deep learning community; the third community is about discourse and sentiment processing; the fourth community is about knowledge engineering, ontologies, and the like; the fifth community is more challenging to characterize, it is probably about pre-deep learning machine learning; and the sixth, and most minor, community is about syntax parsing, similarity, and embeddings.

We leave it as an open question to the reader to attach temporal information to keywords and colorize the communities with respect to the period of use of each keyword.

Hints for Exercise 13-2: Find a Cornelian play among nine Racinian plays

We will collect numeric data for the ten plays and use a method called *Principal Component Analysis* (PCA). The idea of this method is as follows. We will study 24 different (numerical) aspects of the ten plays so that, for us, every play will be a point in $\mathbb{R}^{24}$. PCA will project this cloud of ten points on the plane so that the mutual distances between the points will be globally maximal. By doing this, we hope that some point will be visually separated from the others, and we can then assume that it is the Cornelian play. But PCA doesn't stop there. It also displays vectors corresponding to the dimensions of $\mathbb{R}^{24}$ to see what dimensions contribute to the separation between the points and what dimensions are irrelevant to it.

Here are the variables we will use:

1. verb tenses: present, imperfect, simple past, conditional present, subjunctive present, subjunctive imperfect, imperative present, present participle, past participle (9 features);
2. punctuation: period, comma, semicolon, colon, question mark, exclamation mark (6 features);
3. rhyme schemes: we can search for an AA rhyme scheme, like in

   Ami, n'y rêve plus; c'est en juger trop b**ien**
   Pour t'oser plaindre encor de n'y comprendre r**ien**.

   or an ABAB rhyme scheme, like in

   À nous chercher même il s'empr**esse**.
   Pour l'enfant qu'elle a mis au j**our**
   Une mère a moins de tendr**esse**.
   Ah! qui peut avec lui partager notre am**our**?

   (2 features). The reader speaking French can read Aroui [3], a very interesting study on the metrics, rhyme, and rhythm in Cornelian and Racinian plays. See also Akama & Baudouin[62];
4. phonetic rhyme schemes. In the previous example, the rhyme was graphemic. We can also investigate rhymes on a phonetic representation of the text. This will allow us to capture cases like

   En adoptant Néron, Claudius par son ch**oix**
   De son fils et du vôtre a confondu les dr**oits**.

   which have different graphemic representations but the same phonetic realization (2 features);
5. the presence of specific interjections. Searching in the morphologically analyzed corpus of all Cornelian and Racinian plays, we found that "Ah," "Oui," "Eh," "Hélas," and "Adieu" are the most common interjections. We may as well use them as features (5 features).

We provide three versions of the corpus:

1. 🗋`french-theater.txt`, the plain text;
2. 🗋`french-theater.treetagger`, the previous one, morphologically analyzed by *TreeTagger*;
3. 🗋`french-theater.ipa`, a phonetic transcription of the previous one by *espeak*.

Plays are separated by `###`*nn*`###` lines, where *nn* is a number between 00 and 10.

[62] https://rb.gy/u4otv

Expected Results

When processing the data with *espeak*, the verses we used previously to illustrate phonemic but not graphemic rhymes

En adoptant Néron, Claudius par son choix
De son fils et du vôtre a confondu les droits.

have now become

ɑ̃n adɔptˈɑ̃ neʁˈɔ̃ klodjˈys paʁ sɔ̃ ʃwˈa
də sˈɔ̃ fˈis e dy vˈotʁ a kɔ̃fɔ̃dˈy le dʁwˈa

and we can detect the rhyme “wˈa” in IPA.

The result we obtain can be seen in Fig. 16.19. As we see, plays 01–05 and 08 form a

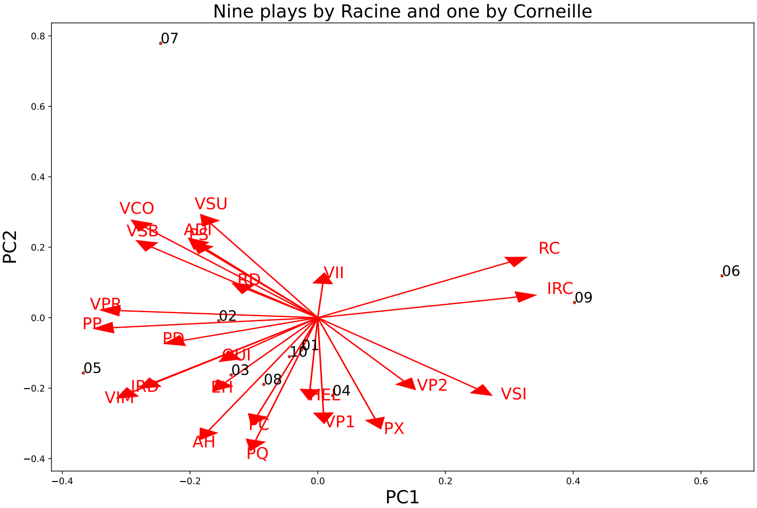


Fig. 16.19 A PCA for detecting a Cornelian play by Corneille hidden among nine Racinian plays

cluster. Play 09 is a bit farther to the right and 06 even more so. And play 07 is placed on the top left. We are facing a Cornelian dilemma (pun!): shall we choose 06 or 07? As the reader may suspect, the plays by Racine and Corneille used in the exercise have been chosen on purpose so that the Cornelian one is as isolated as possible. If we measure the distance of 07 from the nearest point of the cluster, it is greater than the distance between 06 and 09, and between 09 and the cluster. Therefore, the most plausible hypothesis is 07, which is the Cornelian *La Suivante*, a comedy written in 1634 (five years before Racine was born), while 06 is the Racinian *Esther* (1689), and 09 is *Athalie* (1691), Racine’s last work. The other tragedies are: 01 *Mithridate*, 02 *Britannicus*, 03 *Bérénice*, 04 *Iphigénie*, 05 *Andromaque*, 08 Bajazet and 10, his famous *Phèdre*. From the diagram, we can deduce that 07 has more verbs in the subjunctive present and the conditional present, as well as more semicolons, while plays in the cluster have more verbs in participle form and more question and exclamation marks.

The reader interested in the global situation of Racinian and Cornelian works can consult Fig. 16.20, where we display a PCA on the same variables but for the entire Racinian (12

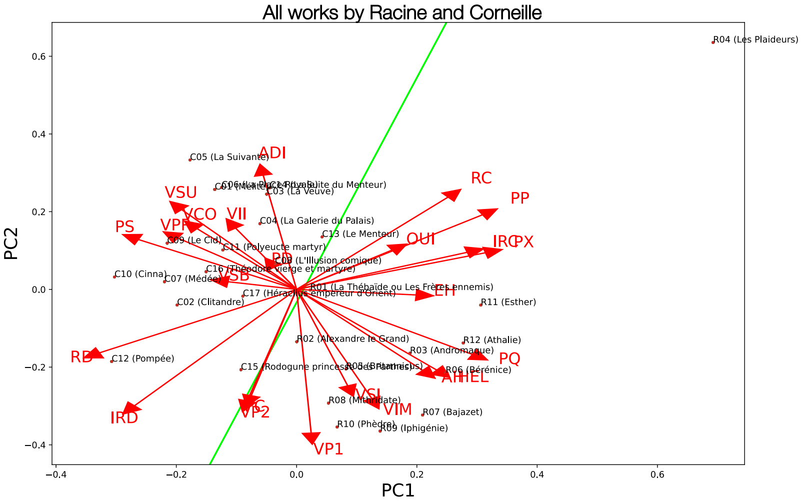


Fig. 16.20 A PCA applied to all Cornelian and Racinian plays. The green line separates the two classes.

plays) and Cornelian (17 plays) corpus. We recognize the cluster between *Mithridate* and *Bérénice*. *Athalie* seems to be part of the cluster, and *Esther* is only a bit higher. On the upper left side, *La Suivante* is the Cornelian play that is the farthest away from the rest. The green line in the middle separates the two classes (Racine on the right, Corneille on the left), and the plays that are closest to the separator (its *support points*) are the Cornelian *Rodogune* and *Le Menteur*, and the Racinian *La Thébaïde*.

Hints for Exercise 13-3: Test properties of GloVe embeddings

The GloVe project page,[63] states that specific linguistic properties become parallel vectors in GloVe's affine space. The properties mentioned are masculine/feminine nouns, company/CEO, city/ZIP code, and comparative/superlative adjectives. In the original GloVe paper [25], GloVe is evaluated by *word analogies* of the kind "*a* is to *b* as *c* is to __." Word analogies are far more general and encompass many linguistic properties.

When one examines the evaluation code on GitHub,[64] one discovers specific word pairs of various kinds, including masculine/feminine nouns. The latter are not called "masculine/feminine nouns," probably because of the inherent lack of political correctness. Instead, the corresponding file is called `family.txt`, and it contains 21 word pairs (not all being really "family," since there is also a "policeman policewoman" pair) and two plural forms. We deduce from this file that the masculine/feminine noun evaluation of GloVe relies only on 23 word pairs. To go further, we have gathered 103 word pairs and their plurals, raising the number of test pairs to 206 (file `masc-femi.txt`). These word pairs were not easy to find since many of them are politically incorrect (such as "steward stewardess" or "milkman milkmaid")—our purpose in collecting them was not to revive their use but to evaluate GloVe.

[63] https://nlp.stanford.edu/projects/glove/

[64] https://github.com/stanfordnlp/GloVe/tree/master/eval/python

As for -onyms, we extracted 148,838 hyperonyms (`hyperonyms.txt`) and 4,482 antonyms (`antonyms.txt`) from WordNet. As some of them are very rare (e.g., "zygaena" is a hyponym of "hammerhead"), we will use word frequencies, as in Exercise 1.1.

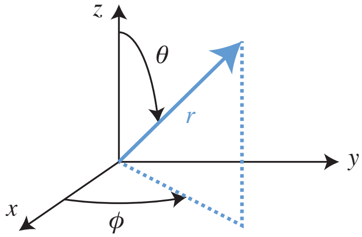


Fig. 16.21 Spherical coordinates (r, θ, ϕ) where r is the Euclidean length of the vector

The statement of the exercise mentioned "hyperspherical coordinates." In Fig. 16.21, the reader can see *spherical coordinates* as used in math, astronomy, geography, and the like. The idea is that the first coordinate, called r, is coordinate-system independent: it is simply the length of the vector (and therefore is always positive). The two other coordinates θ and ϕ are angles of the vector or of its projections with the axes of the Cartesian system (in geography, θ is *latitude*, and ϕ is *longitude*). Spherical coordinates live in $\mathbb{R}^3$. The version of GloVe we will use lives in $\mathbb{R}^{300}$. We will generalize the principle of spherical coordinates and consider that a length and 299 angles represent any word.

The advantage of this approach is that parallelism between two vectors can be considered as having equal angles. Humans perceive angles well (talking about a 5° or 10° angle is a reasonably intuitive physical quantity). We will, therefore, measure the difference of angles between word pairs. Every such difference is a set of 299 angles. We can take their mean and standard deviation and see how well they behave.

This will allow us to evaluate GloVe slightly differently than the evaluation that is part of the GloVe package. We hope it will give a more intuitive feeling of the parallelism of vectors in 300-dimensional space.

The next step is calculating a mean vector for each property (feminization, hyperonymy, antonymy). The choice of the mean vector is essential. This is the vector we will apply to every left member of a pair to obtain the right member of the pair.

We will evaluate two different aspects of GloVe. The first is the stability of the vectors obtained from pairs. Out of these, we will calculate the mean vector of each property. The second aspect we will evaluate is prediction: when we apply the mean vector to the left word, we get a point in 300-dimensional space. The chances that this point is a word vector are zero. The question is: Is the word we are looking for the nearest neighbor of this point in space? And if not *the* nearest neighbor, is it at least among the nearest neighbors? We will calculate the ten nearest neighbors, rank them by proximity to the abstract point, and see how many times the word we expect is first of the list, 2nd to 10th of the list, or is not at all in the list (in which case, we can consider the prediction as a failure).

Expected Results

When calculating hyperspherical coordinates, the results we obtain are of three types:

1. the weighted average and standard deviation of the lengths of the individual word vectors;

2. the weighted average and standard deviation of the lengths of the difference vectors between the word vectors of the pair elements;
3. the weighted average and standard deviation of the mean angles of the difference vectors between the word vectors of the pair elements.

Here are the results:

	(1)		(2)		(3)	
Feminization	6.35	0.46	0.51	0.17	0.07	0.01
Hyperonyms	6.04	0.65	0.72	0.58	0.11	0.02
Antonyms	5.93	0.61	0.69	0.57	0.10	0.02

The first conclusion is that the lengths of the hyperonym and antonym difference vectors are pretty unstable: we have a mean of 0.72 (resp. 0.69) and a standard deviation of 0.58 (resp. 0.57), which is almost as much. A second conclusion is that the feminization difference angles behave pretty well: we have a mean difference of angles of 0.07 radian, which is 4°, a reasonable divergence, with a standard deviation of 0.5°. In the case of hyperonyms and antonyms, the mean difference is a bit higher: 6.3° (resp. 5.7°) with a standard deviation of 1°. Overall, the weighted results are pretty encouraging, probably because the frequent word pairs behave well, and the unfrequent ones do not contribute significantly to the result. We also note that the word vectors we considered have an average length of approximately 6, which is around 1/17.3 of the longest possible vector. This comes from the fact that we are not really in $\mathbb{R}^{300}$ but rather in $[-3.0639, 3.0639]^{300}$ since 3.0639 is the largest coordinate we encounter in (our reduced version of) GloVe. It occurs in the word "thousands." And the length of the diagonal of a 300-dimensional hypercube of edge 2×3.0639 is $\sqrt{300}\times2\times3.0639$, which is 106.14.

To evaluate predictions, we applied the same weighting scheme to Cartesian coordinates. We verify whether the expected word is included among the ten first words in order of distance from the abstract target. Here are the results we obtained:

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	not in list
Feminization	50.0%	6.98%	1.16%	1.16%	0.0%	0.0%	1.16%	3.49%	0.0%	0.0%	36.05%
Hyperonyms	0.79%	0.71%	0.53%	0.43%	0.34%	0.28%	0.26%	0.24%	0.24%	0.19%	95.99%
Antonyms	6.63%	3.03%	1.15%	1.56%	1.31%	0.90%	0.74%	1.06%	0.66%	0.74%	82.24%

We see that for the masculine/feminine word pairs, the odds are reasonable: in 50% of the cases, we get the expected word at 1st position, and in 64% of cases, it is among the ten first candidates. In the other two cases, the results are disastrous: in 95.99% (resp 82.24%) of cases, the word we expect is not even on the list.

We can conclude that the parallelism of difference vectors is relatively OK, and that GloVe could potentially be used for feminization. Still, it can definitely not be used for predicting hyperonyms or antonyms.

References

[1] E. Ali Akbas. "A New Text Steganography Method By Using Non-Printing Unicode Characters". In: *Eng. & Tech. Journal* 28 (2010), pp. 72–83.

[2] Deborah Anderson et al., eds. *The Unicode Standard, Version 15.0*. Unicode Consortium, 2022.

[3] Jean-Louis Aroui. “Mètre. rime et rythme chez Corneille et Racine. Gros plan sur une thèse recente”. In: *Cahiers du Centre d’Études Métriques* 5 (2007). https://crisco4.unicaen.fr/verlaine/ressources/CEM/Beaudouin.pdf, pp. 83–105.

[4] Satanjeev Banerjee and Ted Pedersen. “An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet”. In: *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander Gelbukh. Springer, 2002, pp. 136–145.

[5] Khuyagbaatar Batsuren, Gábor Bella, and Fausto Giunchiglia. “CogNet: A Large-Scale Cognate Database”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3136–3145.

[6] Francis Bond and Ryan Foster. “Linking and extending an open multilingual WordNet”. In: *51st Annual Meeting of the Association for Computational Linguistics: ACL-2013*. 2013, pp. 1352–1362.

[7] *Convention on Road Traffic*. United Nations Economic Commission for Europe, 2006.

[8] John Woldemar Cowan. *The Complete Lojban Language*. A Logical Language Group Publication, 2016.

[9] S. A. Crossley et al. “A large-scaled corpus for assessing text readability”. In: *Behavior Research Methods* 55 (2023), pp. 491–507.

[10] S. A. Crossley et al. “The CommonLit Ease of Readability (CLEAR) Corpus”. In: *Proceedings of the 14th International Conference on Educational Data Mining (EDM)*. 2021.

[11] Johann Nepomuk David. *Das Wohltemperierte Klavier. Der Versuch einer Synopsis*. Vandenhoeck u. Ruprecht, 1962.

[12] Everett Decker. *haiku Emily!* Small Batch Books, 2011.

[13] Gail Fine. *Essays in Ancient Epistemology*. Oxford University Press, 2021.

[14] Norbert E. Fuchs and Uta Schwertel. “Reasoning in Attempto Controlled English”. In: *Principles and Practice of Semantic Web Reasoning*. Ed. by François Bry, Nicola Henze, and Jan Małuszyński. Vol. 2901. Springer LNCS. Springer, 2003, pp. 174–188.

[15] Richard Gillam. *Unicode Demystified: A Practical Programmer’s Guide to the Encoding Standard*. Addison-Wesley, 2002.

[16] Jack Halpern and Jouni Kerman. “The pitfalls and complexities of Chinese to Chinese conversion”. In: *Proceedings of Machine Translation Summit VII*. https://aclanthology.org/1999.mtsummit-1.68.pdf. 1999, pp. 458–466.

[17] Yannis Haralambous and Joseph Dichy. “Graphemic Methods for Gender-Neutral Writing”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 41–89.

[18] Yannis Haralambous and Martin Dürst. “Unicode from a Linguistic Point of View”. In: *Proceedings of Graphemics in the 21st Century, Brest 2018*. Ed. by Yannis Haralambous. Brest: Fluxus Editions, 2019, pp. 167–183.

[19] Hermann Keller. *The Well-Tempered Clavier by Johann Sebastian Bach*. W. W. Norton & Company, 1976.

[20] Franklin Mark Liang. “Word Hy-phen-a-tion by Comp-put-er”. https://www.tug.org/docs/liang/liang-thesis.pdf. PhD thesis. Stanford University, 1983.

[21] Han-Teng Liao. “Conflict and consensus in the Chinese version of Wikipedia”. In: *IEEE Technology and Society Magazine* 28.2 (2009), pp. 49–56.

[22] Tomohiko Morioka. “Viewpoints on the Structural Description of Chinese Characters”. In: *Proceedings of Grapholinguistics in the 21st Century, 2020*. Ed. by Yannis Haralambous. Vol. 5. Grapholinguistics and Its Applications. Brest: Fluxus Editions, 2020, pp. 683–712.

[23] Pierre M. Nugues. *Language Processing with Perl and Prolog. Theories, Implementation, and Application*. 2nd ed. Cognitive Technologies. Springer, 2014.

[24] José-Gregorio Parada-Ramirez. “Les *Voyages extraordinaires* sous le regard des statistiques lexicales”. https://www.academia.edu/50296904/. 2021.

[25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[26] Benoît Sagot. "The Lefff, a Freely Available and Large-coverage Morphological and Syntactic Lexicon for French". In: *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA), 2010.

[27] Tanel Tammet. "GKC: A Reasoning System for Large Knowledge Bases". In: *Automated Deduction – CADE 27*. Ed. by Pascal Fontaine. Springer, 2019, pp. 538–549.

[28] "Tonic Chord, Bach: Prelude and Fugue No.1 in C major, BWV 846 Analysis". https://tonic-chord.com/bach-prelude-and-fugue-no-1-in-c-major-bwv-846-analysis/. 2018.

[29] Shan Wang and Francis Bond. "Building the Chinese Open Wordnet (COW): Starting from Core Synsets". In: *Sixth International Joint Conference on Natural Language Processing*. 2013, pp. 10–18.

[30] Egon Wellesz. *A history of Byzantine music and hymnography*. Clarendon Press, 1961.

[31] John Woods. *The Death of Argument. Fallacies in Agent-Based Reasoning*. Vol. 32. Applied Logic Series. Springer, 2004.

Acronyms

ACE Attempto Controlled English, 201
ACL Association for Computational Linguistics, 423
AI Artificial Intelligence, 2
ALA-LC American Library Association / Library of Congress, 60
ASP Answer Set Programming, 200
ATypI Association Typographique Internationale, 59
AVM Attribute-Value Matrix, 107
AWO African Wildlife Ontology, 300

BCG Basic Conceptual Graph, 331
BERT Bidirectional Encoder Representations from Transformers, 449
BFS Breadth-First Search, 210
BMP Basic Multilingual Plane, 345
BWV Bach-Werke-Verzeichnis, 386

CCG Combinatory Categorial Grammar, 111
CDL Controlled Dalek Language, 483
CDL Character Description Language, 383
CFE Caterpillar Fundamental English, 200
CFG Context-Free Grammar, 97
CG Conceptual Graph, 330
CJKV Chinese Japanese Korean Vietnamese, 351
CL Combinatory Logic, 111
CLL The Complete Lojban Language, 472
CLOI Common-sense Law Of Inertia, 291
CNF Conjunctive Normal Form, 288
CNL Controlled Natural Language, 197
CODL Concise Oxford Dictionary of Linguistics, 1
COW Chinese Open WordNet, 487

DCG Definite Clause Grammar, 496
DFS Depth-First Search, 210

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4

DL Description Logics, 298
DOM Document-Object Model, 369
DRS Discourse Representation Structure, 178
DRT Discourse Representation Theory, 177
DTD Document Type Definition, 311, 364

FCA Formal Concept Analysis, 140
FIFO First-In First-Out, 211
FoaF Friend of a Friend, 314
FOL First-Order Logic, 277
FS Functional-Style notation, 319
FSA Finite-State Automaton, 257

GML Generalized Markup Language, 311
GNU GNU is Not Unix, 251
GPT Generative Pre-trained Transformer, 450
GWA Global WordNet Association, 143

h2g2 The Hitchhiker's Guide to the Galaxy, 217
HAS High Amplitude Sucking, 33
HPSG Head-driven Phrase Structure Grammar, 107

IANA Internet Assigned Numbers Authority, 349
ICM Idealized Cognitive Model, 162
ICTVC International Conference on Typography and Visual Communication, 59
IDC Ideographic Description Character, 461
IDS Ideographic Description Sequence, 462
IEEE Institute of Electrical and Electronics Engineers, 249
IPA International Phonetic Alphabet, 15
IRI Internationalized Resource Identifier, 314

KB Knowledge Base, 286

LEB Life Expectancy at Birth, 217
LMF Lexical Markup Framework, 381
LSA Explicit Semantic Analysis, 413
LSA Latent Semantic Analysis, 410
LSTM Long-Short Term Memory, 439

MAD Michigan Algorithm Decoder, 254
MND A Midsummer Night's Dream, 229
MW Merriam-Webster's Dictionary of English, 1

NFC Normalization Form C, 347
NFD Normalization Form D, 347
NLP Natural Language Processing, vii
NSM Natural Semantic Metalanguage, 147

OCR Optical Character Recognition, 8
OMW Open Multilingual WordNet, 144
OSV Object-Subject-Verb word order, 95
OVS Object-Verb-Subject word order, 95
OWL Web Ontology Language, 319

PCA Principal Component Analysis, 517
PCRE Perl-Compatible Regular Expressions, 251
PENG Processable English, 200
PENS Precision Expressiveness Naturalness Simplicity, 198
POS Part of Speech, 66
POSIX Portable Operating System Interface, 250
PPP Preliminary Phrase Packager, 126
PRS Proof Representation Structure, 202
PUA Private-Use Area, 351

RACE Reasoning in Attempto Controlled English, 497
RDF Resource Description Framework, 315
RNN Recurrent Neural Network, 438
RST Rhetorical Structure Theory, 175

SAI Subject-Auxiliary Inversion, 101
SAX Simple API for XML, 367
SGML Standard Generalized Markup Language, 311
SLIP Symmetric List Processor, 254
SPARQL SPARQL Protocol and RDF Query Language, 316
SSS Sentence Structure Supervisor, 127
SUD Surface-syntactic Universal Dependencies, 126
SVD Singular Value Decomposition, 410
SVO Subject-Verb-Object word order, 95

TEI Text Encoding Initiative, 369
tfidf term frequency inverse document frequency, 397
TOT Tip Of the Tongue, 80
TPTP Thousands of Problems for Theorem Provers, 308

UD Universal Dependencies, 67
UDHR Universal Declaration of Human Rights, 354
UPOS Universal Part of Speech, 67
URI Universal Resource Identifier, 313
URL Universal Resource Locator, 313
UTF-8 Unicode Transformation Format-8, 346

VSO Verb-Subject-Object word order, 95

XML eXtensible Markup Language, 359

ZWJ Zero-Width Joiner, 348
ZWNJ Zero-Width Non-Joiner, 348

Index

Y. Haralambous, *A Course in Natural Language Processing*,
https://doi.org/10.1007/978-3-031-27226-4