NATURAL LANGUAGE PROCESSING AS A COMPUTATIONAL PROBLEM*

Sergio Balari Ravera
Universitat Autònoma de Barcelona
Sergi.Balari@uab.es

1 INTRODUCTION

In the last ten or fifteen years we have witnessed a renewed interest in the development of language technologies as one of the main building blocks of what some visionaries have termed tomorrow's Information Age (see, for example, Dertouzos 1997).

As human-machine interactions become increasingly frequent every day, any step in the direction of making these interactions more fluid will be a step forward in the construction of this new world of the information age. Of course, the obvious way to gain fluidity and smoothness in human-machine interactions is to make them as natural as possible, and what is more natural for a human being than using language for communication? So the argument goes. Leaving aside a number of issues concerning human communicative actions and the role that a number of extralinguistic factors play in this behavior,[1] it is certainly true that the only way of coming close to the world depicted by some science fiction writers (think of Arthur Clarke's HAL 9000 or Douglas Adams' Babel Fish) is to be able to communicate with artificial entities by natural means.

Insofar as this is a common enterprise in which many different scientific communities participate, with distinct views and perspectives, I believe we can identify two different (one would expect that complementary) ways to look at the problem of human-machine communication. On one side, we find an engineering, application-driven approach which, being application-driven, is mainly interested in building systems that work and that are able to provide some efficient, even if limited, service to users. This approach is characterized by taking what could be called a divide-and-conquer strategy, focusing on very specific problems with a clear industrial or commercial application and finding a solution for them. A good (and successful) example of this is the VERBMOBIL project, in which a number of German public and private institutions have been involved in the last eight years and which closed its phase II at the end of July 2000. The system is a speech-to-speech translation device designed as an aid for two individuals interacting in different languages with the goal of fixing a date for a meeting.[2] As the reader may immediately notice, VERBMOBIL, with all its success, is nothing but a grain of sand in the road leading us to the Babel Fish. The point is that this system is not an all-purpose speech-to-speech translation system—something that no one has been able to do so far and that is probably not forthcoming in the next future, but a system that works within an extremely restricted universe (that of meeting dates) that represents a tiny part of many (infinite?) communicative situations in which humans may participate. That VERBMOBIL works is certainly good news, but it is important no to forget that it works precisely because it was

* This paper is a revised and slightly expanded version of a talk delivered at the Summer School on Language Technologies at the Universidad Internacional Menéndez Pelayo in Barcelona. I wish to thank the organizers, Toni Badia and Marsal Gavaldà, for giving me the opportunity to participate in the course. Raquel Fernández read a first draft of the paper and made some helpful comments and suggestions. All remaining errors are my own. The work reported here was partially funded by research grants PB96-1199-C04-01 from the Spanish Ministerio de Educación y Ciencia, and 1999SGR/00113 and CREL 9 from the Generalitat de Catalunya.

[1] The issues are too complex and space is too meager for even attempting a review, but just to make the point, in face-to-face interactions, human communicative behavior goes well beyond speech and is full of visual cues and gestures with specific functions that help making the interaction more fluid (see Hauser 1996 for discussion and references). This example alone may be helpful in providing an idea of the dimensions of the problem of modeling communicative behavior.

[2] Further information about VERBMOBIL may be found in its home page at http://verbmobil.dfki.de/ and in a recently published monograph, Wahlster, ed. (2000).

designed to work within the limited universe in which it works. It is important because, as M. García Carpintero pointed out in the round table held during the closing session of the Workshop, the flashing success exhibited by the engineering approach in some respects may easily hide the real state of our current knowledge about how human language really works and may create some false expectations about the future of language technologies. As this has already happened and the outcome was the ALPAC report of 1964 with its devastating consequences for the field, it is wise to take García Carpintero's advise seriously.[3]

But what can we do, then? My contention in this paper will be that, with all the success of the engineering approach notwithstanding, a parallel and much more long term oriented approach must be pursued: what I call here the computational approach. I could also have used the words 'cognitive modeling', as this approach is based on the premise that a possible solution for the problems I've been discussing is to be able to reproduce the 'real thing' by artificial means, and, so far, the 'real thing' is the collection of human cognitive capacities that participate in communicative behavior.[4]

My plan for the rest of this paper is to give a rough-and-ready picture of how we can look at natural language processing from this perspective by discussing, first, what I believe is the central tenet of the computational approach: the distinction between a theory of representations and a theory of processes. As I go on, I will try to provide examples of how these theories interact and how they can influence each other. A cautionary note is at stake before I go on, however: Cognitive Science is an extremely heterogeneous discipline where converge (and often diverge) the interests and methods of psychology, computer science, linguistics, philosophy and neuroscience (to name the disciplines most clearly involved in the study of human cognition), but within which many disparate theoretical positions are held. Thus, it will almost certainly be the case that many cognitive scientists disagree with much of what I will say here.

2 THE COMPUTATIONAL PERSPECTIVE

As the title to this paper suggests, my intention here is to look at the problem of Natural Language Processing (henceforth NLP) from a computational perspective. In other words, the NLP problem, whatever our ultimate goals are, is better approached from a computational perspective and with computational means. But, what it means for something to be computational? Well, the answer to this question is much less obvious than you might expect. The point is that if you think that an answer to this question necessarily involves making some reference to computers, then, you are wrong. True, computers are devices for performing computations (whatever that may be), but they are not the only computational devices: brains are computational devices, abacuses are computational devices, and your hand, armed with pencil and paper is a computational device too. A nice property of computational processes is that we can look at them from a perspective that is totally independent from the device that actually implements them. This is not to say that implementation is irrelevant, but just that we can go a long way learning

[3]For a quick review of the history of the field, the reader may like to read Wilks & Sparck Jones (1983). The paper is also interesting because, even if written in 1983, many points that it makes are still applicable today in the year 2000. Another illustrative paper is Dennett (1998; originally 1985) together with is Postscripts.

[4]This is not to say that the goal of Cognitive Science is to simulate human behavior artificially (although it will be an error to despise the enormous possibilities of computer models in this kind of research; see the papers in Langton et al, eds. 1992 for some striking examples), as the goal of understanding and explaining the mysteries of human cognition is an enormous and enticing scientific enterprise in itself. My point is that, in the long run, one may derive more benefits than losses by also pursuing the more theoretical avenue of the cognitive sciences. In fact, I believe that most human and social sciences will benefit from taking the cognitive approach seriously. Some, like archaeology, appear to have gotten the message (see Renfrew & Zubrow, eds. 1994), others, like economics, remain in cognitive oblivion (with notable exceptions, see Frank 1988).

about a computational problem without taking this particular matter into account; this is one of the most important elements of the intellectual legacy of the British mathematician Alan Turing, and it is good not to forget it.

But let's go back to our question, what it means for something to be computational? Here, I will need some special help, so let me quote the following lines from a book by the late cognitive scientist David Marr (1982, pp. 24-25):[5]

«[T]he different levels at which an **information processing** device must be understood [are:] At one extreme, the top level, is the abstract computational theory of the device, in which the performance of the device is characterized as a **mapping** from one kind of information to another, the abstract properties of this mapping are defined precisely, and its appropriateness and adequacy for the task at hand are demonstrated. In the center is the choice of **representation** for the input and output and the **algorithm** to be used to transform one into the other. And at the other extreme are the details of how the algorithm and representation are realized physically—the detailed computer architecture, so to speak.»

I've highlighted a number of terms in this quote, which will help us define the thread we will be following from now on. Marr was primarily concerned with modeling human vision, but, and here you see the beauty of Turing's ideas at work, this is totally irrelevant, as one of the advantages of taking a computational perspective is that we have a general framework for looking at several different things from this view. OK, then, now we have some fairly important points to make about NLP, namely that:

- It can be defined as an **information processing** task to be characterized as a **mapping** from an input **representation** to an output **representation**, such that the mapping may be described by means of a specific **algorithm**.

I left the highlights on the keywords because, leaving aside for the moment the fact that NLP is an information processing task, I want to spend some time discussing the relevance of the other elements and analyzing the relations that hold between them. Let me start with an analogy that may be of some use presently. Consider an artistic activity like sculpture. In a preliminary stage of producing a sculpture, the author designs it project as an abstract mapping from some (yet to be determined) lump of matter and a specific form that only exists in the mind of the sculptor and/or as a series of sketches on paper. In the second stage, the author needs to decide what type of material is she going to use. There are several considerations that do not concern us here (e.g., aesthetic ones or considerations relating to preferences and skills of the sculptor), the point is that, whether the sculpture is going to be made of stone, wood or metal, this decision determines the production process in the sense that every material must be treated with the appropriate techniques (i.e., you can carve wood, but you cannot melt it); and the other way round, if our sculptor prefers to use some specific technique, she won't be able to use any kind of material.

The point with this analogy and that I expect to make clearer with the following example is that, in many cases, the nature of the entities that participate in the process determines the way the process is to be performed, and vice versa, if we want to do something in a specific way, we will be putting a constraint on the nature of the entities we will be able to manipulate. Translated into computational terms, this analogy may be restated as follows: choice of a mode of representation will condition the choice of an algorithm, and vice versa. Let me give now another example, which is closer to the situation I want to describe.

[5]Marr's definition of the three levels is useful to start the discussion, this is not to say that it is without problems and that others, probably more accurate definitions are possible; see Jackendoff (1987, Ch. 4) and Peacocke (1986), for discussion of this particular topic.

This is a story about numbers. It's a long time humans know about numbers, about their properties and applications. In order to be able to use them, humans have had the need to represent numbers somehow. As far as I know, nobody has ever seen a number, they could be fluffy and pink or bubbly and green, who cares? What is important is that, to use them, we need to represent them; as long as the mode of representation we choose preserves the relevant properties of numbers that allow us to do such things as addition and multiplication, then, no problem.

Greeks and Romans where the first western cultures to discover many of the properties and applications of numbers. Greeks, for example, discovered irrational numbers, such as π and $\sqrt{2}$ (irrational numbers are not mad numbers, but just those that cannot be expressed as the ratio of two integers, like 1/2). That this is so is quite surprising (and, in a way, it says a lot about the skills of Greek and Roman mathematicians and engineers), as they both were using a fairly odd way of representing numbers. Let me remind you how the Roman system worked.

The system is based on a collection of seven symbols representing selected numbers, plus a few rules for combining these symbols in order to construct more complex ones.

BASIC SYMBOLS FOR ROMAN NUMERALS	
Symbol	Number represented
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

As for the principles of combination, you all know how they are approximately supposed to work: a) A symbol to the right of another symbol has an additive function; b) A symbol to left of another symbol has a subtractive function. With this system of representation, performing addition and subtraction is fairly straightforward, since the very idea of these operations is directly built in the system of representation. (In fact, you can think of Roman complex numerals as a way of writing sums, with implicit addition and subtraction signs connecting the symbols.) But what about the algorithms for multiplication and division? Well, multiplication is a nightmare; division is just impossible. To appreciate this, consider the simple situation of multiplying 32 by 46:

XXXII times XLVI is the same as:
X+X+X+I+I times L+V+I–X. We construct the following auxiliary table:

X times L = D	X times I = X
X times V = L	X times X = C

X+X+X times L+V+I = D+D+D+L+L+L+X+X+X
X+X+X times X = C+C+C

Thus,
XXX times XLVI = D+D+D+L+L+L+X+X+X–C–C–C.

Next we calculate I+I times L+V+I–X with the auxiliary table below:

I times L = L	I times I = I
I times V = V	I times X = X

I+I times L+V+I = L+L+V+V+I+I
and
I+I times X = X+X

Thus,
II times VLVI = L+L+V+V+I+I–X–X

Now we just have to add the two results:
D+D+D+L+L+L+L+L+X+X+X+V+V+I+I–C–C–C–X–X =
M+D+C+C–C–C–C+L+X+X+X–X–X+V+V+I+I = MCDLXXII

Now you know why the Roman numeral system only survives to inscribe dates on gravestones or to indicate the copyright year of Hollywood pictures. It's just useless for other practical purposes. If you now think of our current numeral system, the so-called Arabic system, you'll notice that, with the same basic idea of using symbols to represent numbers, it uses a different set of principles to combine these symbols to build more complex representations. The system is a positional one and recognizes a quite useful entity, the number zero. With this system of representation, all four operations are easy to perform, just choose your favorite algorithm for doing multiplication—there are several of them—and you will be able to check that 32x46 is, in fact, 1472.

Back to our topic, we see that there is a morale in this story about numbers, namely that, in characterizing a computational task, we must be very careful at the time of choosing a format for our input and output representations and at the time of choosing an algorithm to describe the process of computing a representation out of another. It is not true that just anything will do, since it is not true that the form of our representations has nothing to do with how they will be processed. Thus, when choosing an appropriate format for our representations we must be careful not to fall into the RNT (short for Roman Numeral Trap), and the only way to avoid that is to know what are the underlying formal properties of our representation language. It's the only way to know which algorithm will be more suitable for the task we have defined or, indeed, whether there is one at all. Let us then summarize the three basic elements of any computational theory:

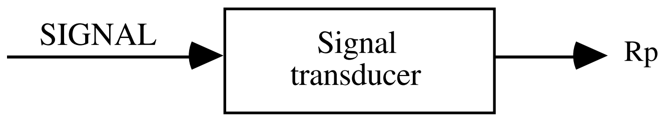
1.- Definition of a mapping from an input representation into an output representation;
2.- A theory of input and output representations, to be modeled by an appropriate representation language;
3.- A theory of how the mapping is actually performed, taking into account the nature of the representations involved.

OK, let's expand a bit the first point in this list, so that we can later look at the other two. Now, to do this we can reformulate point 1 as a question, namely

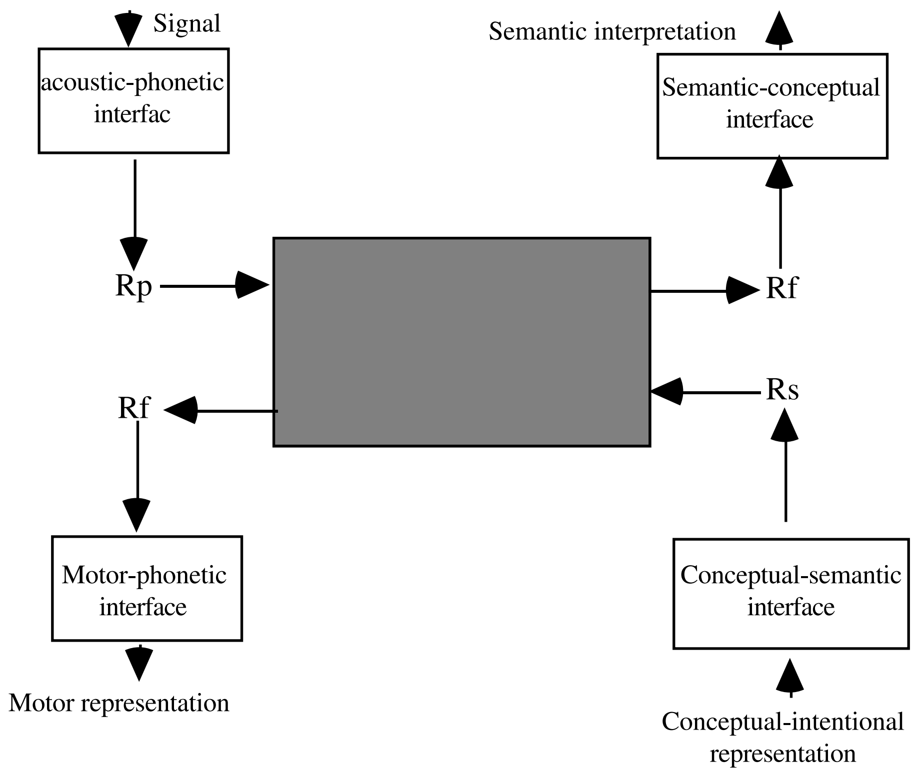
- WHAT SORT OF MAPPING IS INVOLVED IN AN NLP TASK?

And the answer is... well, it depends. It depends because the NLP problem contains, in fact, two subproblems: the Parsing problem and the Generation problem or, to put it into psycholinguistic terms, the Comprehension and the Production problems. Thus, depending on what direction you are going, the mapping will take some specific type of input representation or another and will produce some specific type of output representation.

From the point of view of comprehension/parsing, the input may be either an acoustic signal or some string of characters, and the output we may assume that is a complete structural description out of which a semantic interpretation may be derived. In fact, in the case of the acoustic signal, it may be practical to assume that the it is translated to some normalized phonetic/phonological representation that acts as the actual input to the process:



In the case of production/generation, the input must be a semantic representation of some sort (a logical form, for example) and the output a complete structural description out of which a phonetic interpretation may be derived. The following scheme, adapted from Jackendoff (1997), summarizes this:



This is a very general model, not just a psycholinguistic one; replace the labels with others, less biased ones and you'll get the model for a complete speech comprehension/production system. But I want you to focus your attention on the central box and on the representations getting in and out of it, since these are the mappings we will be mainly concerned with. The point is that we need a theory of how these representations look like and how the mapping is actually performed. Here is where linguistic theory and the theory of computation get in touch. The former will provide us with a grammar in charge of defining the form of a complete representation for a grammatical sentence, whereas the theory of computation will provide us with an algorithm defining the actual operations and steps that we must follow to produce that representation given some specific input information. The tighter point of connection will be at the level of the grammar, to be interpreted as a store of knowledge to be used by the algorithm during the process of constructing the representation. In the next section I deal with this in some detail.

3 GRAMMARS AND PARSING

As a starter, and to get a first idea of how things might work, let's consider a very simple mapping, from strings of characters into phrase structure trees. This means that our 'linguistic theory' will be extremely simple and unable of capturing many phenomena of natural languages, as we will be using a simple Context-Free Phrase Structure Grammar. The reason for doing this is that with such a simple grammar we will be able to exemplify more clearly many things about processing algorithms.[6] Later on we will consider a more linguistically adequate way of writing grammars that can be processed with slight extended versions of the algorithms considered here.

Consider the following Grammar:

1.- S → NP VP	6.- VP → TV NP VP
2.- VP → IV	7.- NP → Det N
3.- VP → IV PP	8.- NP → Det N PP
4.- VP → TV NP	9.- PP → P NP
5.- VP → TV NP PP	
10.- Det → {the, her}	13.- IV → {sneeze}
11.- NP → {her, they, teachers}	14.- TV → {hear, see, read}
12.- N → {teachers, book, report}	15.- P → {on}

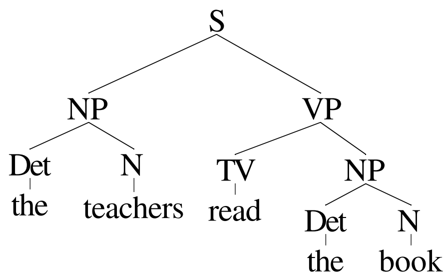
Now we have a toy grammar, complete with a small lexicon. With this grammar, and provided we find the appropriate algorithm, we will be capable of assigning structures to sentences like:

a.- the teachers read the book
b.- the teachers sneeze
c.- they see the book on the teachers
d.- they hear her report on the teachers

Or better still, what our grammar says is that there is a limited set of correct representations for a sentence given the words contained in it. In fact, this grammar, being rather small, allows just for one representation per sentence. For example, by rule 1, it says that all sentences are made up by two immediate constituents, a Noun Phrase (NP) and a Verb Phrase (VP). NPs and VPs may be of different kinds depending on the

[6]The contents of this section is based on Chapter 6 of Gazdar & Mellish (1989).

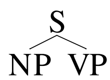
lexical material they contain: rules 2 to 6 give a number of possible structures for VPs, whereas rules 7 and 8 define the possible structures for NPs; rule 9 defines the structure of Prepositional Phrases (PP), those constituents that are headed by a preposition. Finally rules 10 to 15 provide us with information about the categories of words: we have Determiners (Det) like ‘the’, Nouns (N) like ‘book’, Intransitive Verbs (IV) like ‘sneeze’, Transitive Verbs (TV) like ‘read’ and, finally, Prepositions (P) like ‘on’; notice that in the lexicon we also have some word of category NP. Thus, our grammar tells us that, for instance, a sentence like ‘the teachers read the book’ must have the following representation:



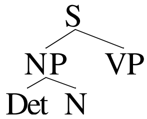
Notice that the grammar, seen independently of the algorithm just tells us how an appropriate representation of a sentence looks like. This is usually what linguists do: they develop theories of how we can represent the structure of natural language expressions to capture their (sometimes very complex) properties. But seen from the perspective of NLP, the grammar does something else, it defines the set of all possible goals of the algorithm; in other words, the grammar defines all possible output representations in our mapping, although it does it in an extremely concise way, as the number of rules in the grammar is usually small, whereas the set it defines may be very large, perhaps infinite. This is a very important property of grammars, since they provide us with the means of using finite resources in the definition of a possibly infinite set of output representations.

Now, we want a procedure that is capable of assigning a structural description to any of the sentences above using the information stored in the grammar; in other words, we want a procedure that, using the finite resources provided by the grammar, is capable of mapping any input string into its appropriate structural description (the output representation) This is essentially what a parser does, and it can do it following two different strategies: top-down or bottom-up. Parsers of the first type are, in some sense, hypothesis driven, that is they start building a structure from the top node and proceed down until they reach a terminal node (i.e., one that contains a word). For example, let’s parse top down the sentence ‘the teachers read the book’:

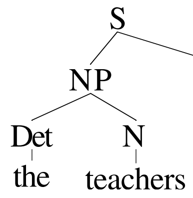
STEP 1: The parser starts expanding rule 1 and builds the first piece of structure with an S dominating an NP and a VP node.



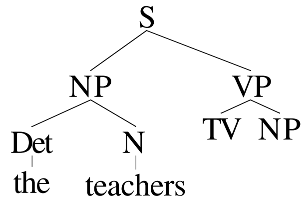
STEP 2: Proceeding left-to-right, the parser has the possibility of expanding the NP node into a terminal, but, since the first word in the input is not an NP, it rejects this hypothesis and expands rule 7 instead (we assume that rules are ordered and that the parser chooses the first appropriate rule):



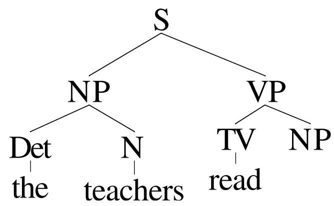
STEPS 3-4: Both Det and N may be expanded as terminals and the construction of the first NP is completed:



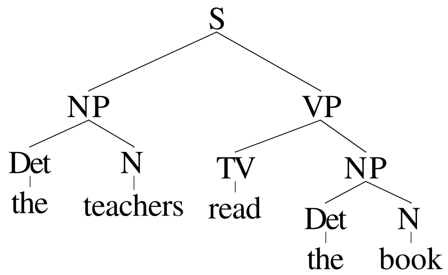
STEPS 5-7: The parser will try, in that order, to expand rules 2, 3 and 4. Since neither rule 2 nor rule 3 contain a transitive verb, these hypotheses, once tried, will be rejected. Here we see the result of expanding rule 4 in STEP 7:



STEP 8: The parser finds the verb.



STEPS 9-10: The parser builds the second NP and, since there are no more words in the input, the process ends:

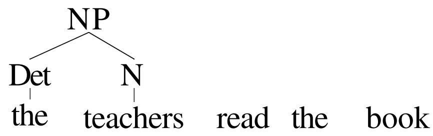


As you see this parser follows what is technically known as a left-to-right depth-first regime, always expanding the leftmost nodes and going as deep as it can until it finds a terminal. Another important point is that it proceeds by try-and-error, making hypotheses that it must verify before building a piece of structure, this means that it may end up building lots of spurious structures before reaching its goal: the more complex the sentence the more hypotheses it may need to verify. As we will see presently, there exist techniques for reducing this search space and to speed up the process. But before considering these, let us first look at how would a bottom-up parser proceed.

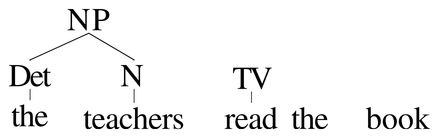
Here the parser looks first at the first word in the input string and expands up the appropriate node. This is STEP 1:

STEP 2: The parser will check whether there is any rule of the form K → Det in order to see if it is possible to build some structure out of that Det. Since there isn't one, it reads the next word in the input and assigns it the corresponding category.

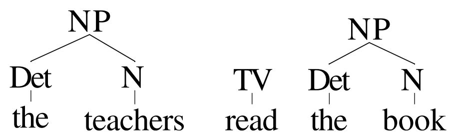
STEP 3: Now, it seeks a rule of the form K → Det N, since rule 6 is exactly of this type, it applies it:



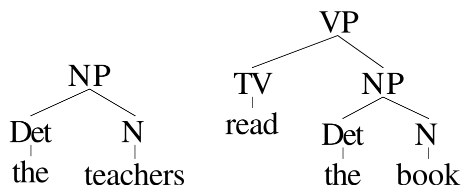
STEP 4: Here, the parser will look for a rule expanding a single NP. There is none, so it reads the next word in the input:



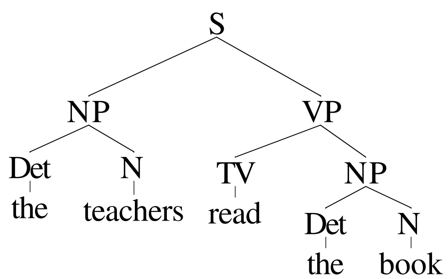
STEPS 5-7: The parser will try to expand the TV, but there are no rules containing just a TV, so it will try to build structure out of the first NP and the TV, since this is not possible either, it will read the next word, which is a Det. Here, exactly the same process for the building of the first NP will be followed. In the end, the result is this:



STEP 8: The parser finds that it can build a VP out of a TV and an NP:

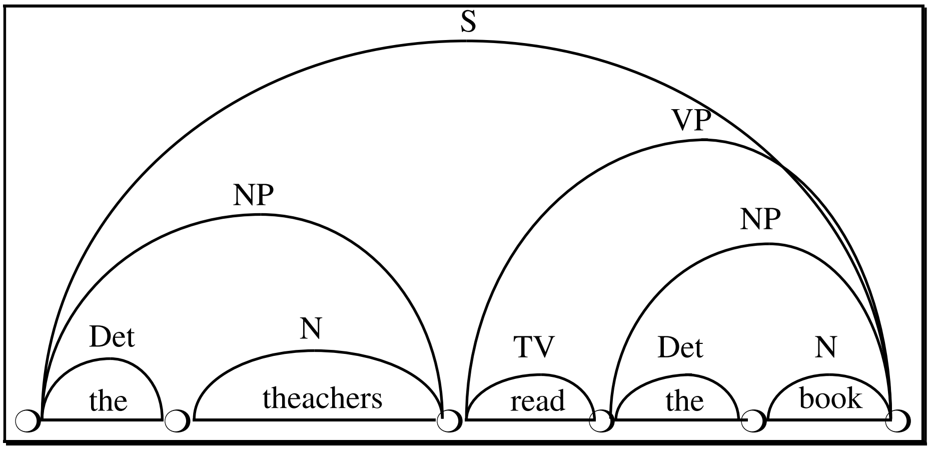


STEP 9: The parser builds an S out of the first NP and the VP:



These toy examples are interesting because they all show a shortcoming of doing things in such a crude way. The fact is that there is a lot of redundancy in the work that the parser is doing, as the same situation pops up again and again. This is so because the parser has no memory of what it has done before, so it may be of some use to provide it with some means for recording the constituents (and their structure) that it has found.

A preliminary answer to this problem is to provide the parser with the means of constructing a well-formed substring table. These are tables where successfully found phrases are recorded. When a phrase of a particular category is then sought at a particular position in the string, the table is consulted. If there is an entry for the right kind of phrase at the right place, then the entries in the table are used as the possible analyses, rather than the work having to be done again. Thus, the finished WFST for the sentence above, will look more or less like the following:



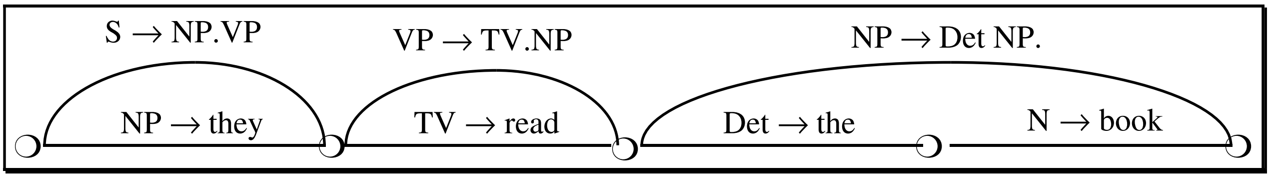
This is already useful because the parser will not have to build the second NP, since it has exactly the same structure as the first. A WFST, however, will not prevent the parser from reinvestigating hypotheses that have previously failed. If we want to avoid duplicating previous attempts at parsing, we need something else: an explicit representation about the different goals and hypotheses that the parser has at any one time. To do this, we need to introduce a few modifications to WFSTs, the most important of which concerns the labels found in arcs. Instead of labeling arcs with a category, we will label them with rules, but with rules annotated in such a way that we will have the information whether the hypothesis has been tested or not and to what extent. Consider these:

a.- S → ●NP VP

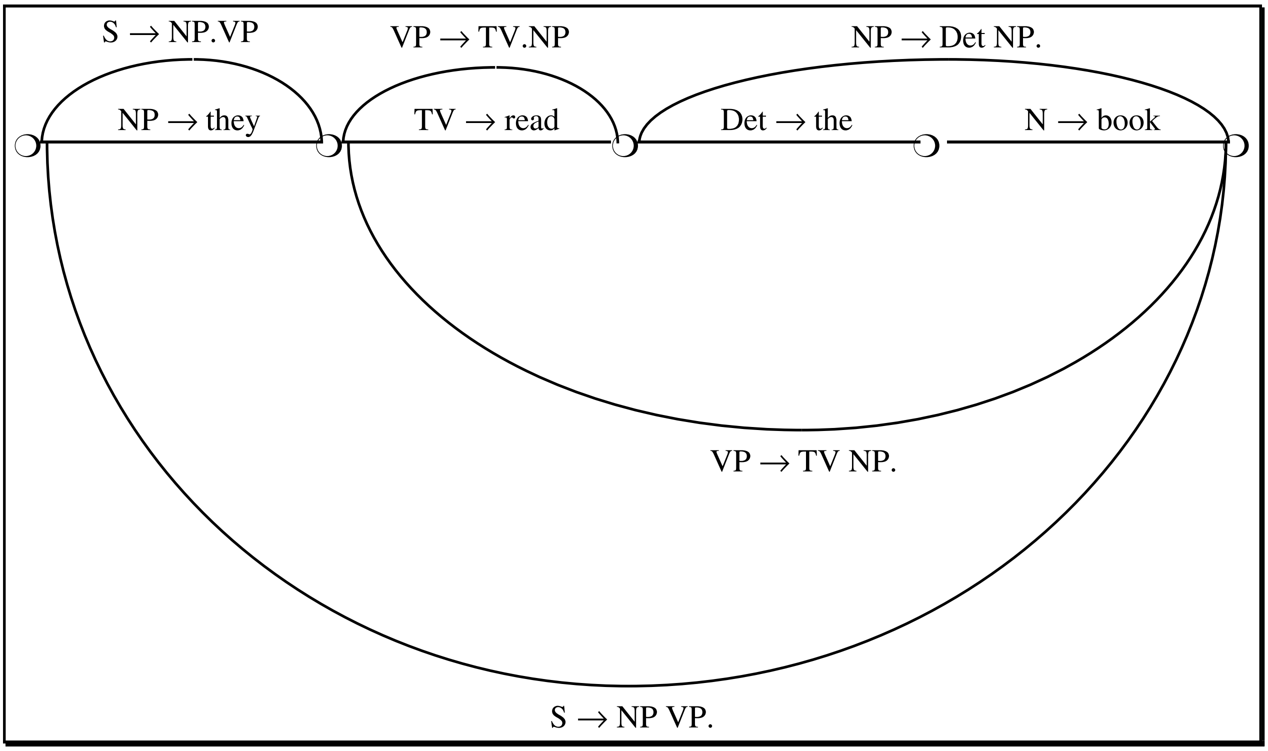
b.- S → NP ●VP

c.- S → NP VP●

The dot in the rule is an indication of the extent to which the hypothesis has been pursued. In the (a) case, if this object is labeling an arc, we know (or, rather, the parser knows) that the hypothesis has been made, but not tested; the (b) case tells us that the hypothesis has only been partially confirmed, whereas the (c) case reflects the case where a hypothesis has been fully confirmed. Thus, as parsing goes along, a chart containing this information is built as a representation of the goals and the hypotheses made by the parser. For example, for the sentence ‘they read the book’, at some point during processing we may have the following active chart:



Now we can see how the chart can save us some time during processing by applying a heuristic rule called The Fundamental Rule of Chart Parsing.[7] What the Fundamental Rule tells us is that if we have an arc (or edge, as they are usually called) labeled A → W1 ●B W2 and an edge labeled B → W3●, then we can add a new edge labeled A → W1 B●W2. Following this principle, our previous chart automatically becomes the following:



There's a lot more to be said about parsing in general and chart parsing in particular, but with this rather impressionistic view, I believe that you can get a fairly good idea of how this is supposed to work. Note, by the way, that we have been able to talk about parsing and parsers without making a single reference to computers or programming: the algorithms and strategies we have reviewed here are completely independent of that and may be implemented in any programming language (in some languages more efficiently than others) and within any machine. Another advantage is that these algorithms are relatively easy to extend to richer systems for representing linguistic knowledge, which is what we now turn to.[8]

---

[7]The idea of chart parsing is usually attributed to Martin Kay, who discusses the issues presented here and some more in Kay (1980).

[8]For a detailed discussion and practical implementation examples of the kinds of grammars I am about to present, the interested reader is referred to Gazdar & Mellish (1989) and to Pereira & Shieber (1987) and references therein.

4 A BETTER GRAMMAR

The parsing techniques described in the previous section were developed and extended during the second half of the 1970s and fostered the hope that Context-Free Phrase Structure Grammars (CFGs) of a similar type as our toy grammar above might be sufficient to capture all the complexity of natural languages. This was soon seen not to be true, at least for CFGs that used atomic symbols in their rules, again like the one we used above. In contemporary linguistic theory there was, however, a fairly strong tradition of using features as a formal tool for expressing certain properties of linguistic objects. Thus, for example, within phonology the standard conception of a phoneme was (and is) that of a non-atomic entity whose internal structure is conceptualized as a bundle of features with the function of expressing the set of properties that distinguish an individual phoneme from the other phonemes in the inventory of a language. Within semantics, lexical decomposition models also have tried to represent the meaning of words as the result of combining a bundle of semantic primitives in such a way that different combinations would yield different lexical meanings. Even syntax had long been using features to represent the internal structure of morphosyntactic categories: almost anybody with some knowledge of linguistics is familiarized with the proposal by Chomsky (1970) of classifying nouns, verbs, adjectives and prepositions according to the [±N, ±V] features.

A shared element of all these approaches is that of using features with a very simple structure. In general, a feature is conceived of as an attribute-value pair, where the attribute expresses some property (e.g., voiced, animate, N) and the value is an atom, usually Boolean, that adds information about the property denoted by the attribute. Therefore, we often get features like [+voiced], [–animate] or [+N] and, less frequently, features like [person=1st], [number=singular], and so on. Already in the mid 1980s, some linguists started to explore the possibility to use recursive (or quasi-recursive) features to extend CFGs, that is to say, features whose value may be a complex feature bundle instead of just an atom. Thus, we start finding descriptions of linguistic object like the following:[9]

$$\begin{bmatrix} \text{category } N & \\ \text{inflection} & \begin{bmatrix} \text{person } 1st \\ \text{number } sg \\ \text{gender } fem \end{bmatrix} \end{bmatrix}$$

This complex feature bundle may roughly be interpreted as follows: "there is a category of type N, which possesses the property of INFLECTION, which includes the morphosyntactic categories of PERSON, NUMBER and GENDER, and the type of person is *1st*, the type of number is *singular* and the type of gender is *feminine*."

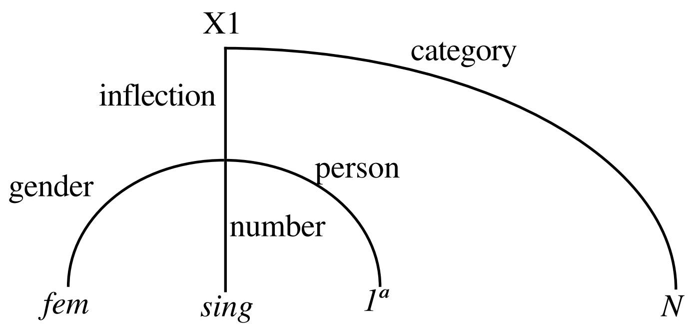
With such a small extension, the expressive power of feature-based systems gets enormously enhanced; it is not strange, therefore, that since the mid 1980s a number of linguistic frameworks were developed that make a more or less extensive use of this notion of complex feature as an alternative to the, at the time, favorite formal device among linguists: transformational rules.[10]

---

[9] The most articulated version of a theory based in extended CFG with the properties described in the text is the GPSG of Gazdar et al. (1985).

[10] All the following linguistic frameworks make extensive use of complex features and eschew transformational rules completely: Lexical Functional Grammar (LFG; see, especially, Kaplan & Bresnan 1982), Generalized Phrase Structure Grammar (GPSG; Gazdar et al. 1985), Head-Driven Phrase Structure Grammar (HPSG; Pollard & Sag 1987 and Pollard & Sag 1994), and a number of varieties of categorial grammar (e.g., Zeevat et al. 1987 and Uszkoreit 1986).

Another shared property of these models is the adoption of feature structures as the mathematical entities used to model linguistic objects. Feature structures are graphs, whose geometry represents the relation existing between an attribute and its value. Thus, we can model the feature bundle above with the following feature structure:



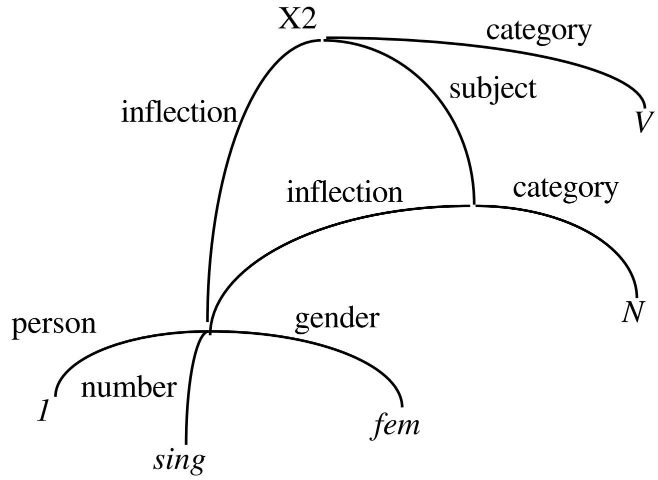
As the reader will notice, the attribute-value relation is expressed, in a feature structure, as a pair <arc, node> such that the label associated to the arc is the name of the attribute and the label associated to the node is the name of the value. In this simplified version, only nodes representing an atomic value bear a label; nodes with complex values are distinguished by the feature substructure stemming from them, which is, in turn, potentially complex.

A typical property of feature structures is that there is always a path from the root node (the one marked with an X above) leading to some node with an atomic value. This means that, by transitivity of the attribute-value relation, we may refer to atomic values as values of paths:

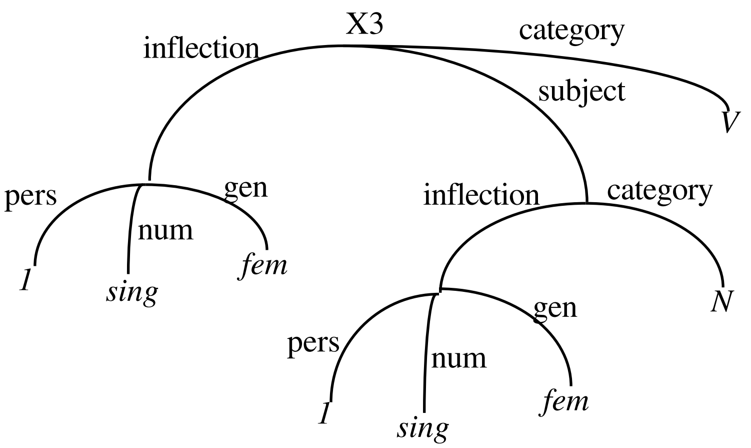
a. X_1|category = *N*.
b. X_1|inflection|person = *1ª*.
c. X_1|inflection|number = *sing*.
d. X_1|inflection|gender = *fem*.

In the example I follow the standard convention of labeling the root node with an X and a subindex and the also standard convention of using a vertical bar to separate attribute names in a path.

Another fundamental property of feature structures is that they have the power to represent *structure sharing* relations. Geometrically speaking, two paths share structure when both converge in the same node and, hence, share exactly the same value. For example, assume that you want to express the agreement relation holding between a subject and a verb. Structure sharing allows us to express this directly in the feature structure that represents the verb:



Abstracting away from the descriptive adequacy of this representation (in English, verbs do not agree in gender with their subjects), we can see how the feature structure above has structure sharing: the paths X_2|inflection and X_2|subject|inflection converge at the same node and from this node stems a substructure representing the inflectional properties shared by noun and verb. It is important to insist on this particular point: we cannot just say that both paths have the same value, or that it is equal or of the same kind; their value is exactly, physically, the same object. According to this, the following representation, without structure sharing, is different from the previous one:



They are different because in the second case the values of the paths X_3|inflection and X_3|subject|inflection are different objects; they may be objects of the same type, but not the same object. This means that, in the world of feature structures, the notion of identity may apply to two different situations, traditionally known as *token identity* and *type* (or *sort*) *identity*, where the first implies the second but not conversely.

The importance of capturing the difference between token and type identity forces us to make explicit an idea that so far was only implicit in the system I've been describing: feature structures are organized in types in such a way that every type of feature structure is distinguished from the other types by the fact that it possesses a number of appropriate attributes for that type and not others, and by the fact that it requires that the values of these appropriate attributes belong to specific types. For example, we could say that the attribute INFLECTION takes values of type *inflection* and that this type has as its appropriate attributes PERSON, NUMBER and GENDER, which, in turn, take values of type *person*, *number* and *gender*, respectively.[11] Of course, types may have subtypes, such that an attribute taking values of type *number*, may select among several available subtypes of *number* like, for example, *singular*, *plural* or *dual*. Notice, then, that the value of an attribute is always a type, regardless of the fact whether it has appropriate attributes or not; those types with no appropriate attributes correspond to what so far we were calling atoms.

As the reader may have noticed, the notion of type is a formal constraint we impose over feature structures. There is nothing in the mathematical definition of feature structure requiring such a constraint. Our motivations are mostly formal—we need to capture the difference between token and type identity, but also linguistic, since we want to avoid situations in which the CATEGORY attribute, for example, takes the type *singular* as its value. In fact, from the linguistic point of view, we want to impose constraints over what is a possible linguistic object, and the formal notion of type appears to be quite useful for that. In the following and final section of this paper we take over all these notions in order to provide a novel view of how to conceive of natural language grammars.

5 UNIFICATION-BASED GRAMMARS

I concluded the preceding section introducing the notion of a type of feature structure and pointing to the formal and linguistic reasons that motivate its adoption. Let's explore some interesting consequences of this move.

First, it is obvious that the notion of type has implicit the idea of an ontology, that is, and staying within the world of linguistics, that there exists a number of entities with well-defined properties, that are part of the structure of language. All linguistic theories have more or less explicit ontologies and when linguists talk about 'nouns', 'verbs', 'empty categories', 'morphemes' or 'passive constructions' they are usually referring to types in an ontology. The idea of an ontology is, therefore, not alien to linguistic theory and, sometimes, some differences between competing theories may be traced to differences in the ontologies assumed.

Pollard & Sag (1987) already saw the importance of building an ontology of linguistic objects (or, to use their term, a sort system) and the need that the phrase 'linguistic object' does not refer only to a full representation but also to substructures within this representation. Thus, for example, we may say that all representations are objects within the class of *signs* because we assume that all representations have something in common, i.e., belong to the same type or are of the same sort. Assume, with Pollard and Sag, that all objects of sort *sign* share the property of having phonological, syntactic and semantic information. Now, any proper part of a sign may itself be conceived of as a linguistic object of some sort, such that we may talk about an intonational phrase (phonological object), a noun phrase (syntactic object) or a generalized quantifier (semantic object) without talking, necessarily, about a different sign. Seen from this perspective, the world of linguistic objects is much, much larger, since it goes well beyond full representations of sentences or constructions, which, in this view, are nothing but a special class of linguistic objects. As we will see presently, a grammar may be defined as a collection of declarations about how these objects can be combined and what their internal structure can be.

Getting deeper into the notion of an ontology and the associated notions of sort and subsort we can obtain a better understanding of the consequences that a formally precise

[11]To avoid confusion I will adopt the convention of writing attribute-names in small caps and type-names in italics.

definition of the sort system has for the theory.[12] Firstly, it is important to note that any sort system is hierarchically organized as a multiple inheritance network, in such a way that there always is a unique maximal or more general sort of which the other sorts are a subsort. Given the relation that holds among the sorts in the system, every sort inherits some property from its supersort. Consider the following sort system, inspired in that of Pollard & Sag (1994):

entities

signs . . .

words *phrases*

As you can see, this ontology has a maximal sort, that of *entities*, which, in turn, has a number of subsorts. I focus here on the subsort of *signs,* which is partitioned into two more subsorts: *words* and *phrases*. Following Pollard and Sag, we can define a *sign* as a linguistic object with phonological, syntactic and semantic information. These properties will be inherited by all subsorts of *sign* and, possibly, some specific ones will be added for every subsort. In other words, a *word* IS A *sign* and a *phrase* IS A *sign*. Sorts and subsorts are connected by the basic ISA relation, which is transitive, such that if a *word* IS A *sign* and a *sign* IS AN *entity*, a *word* IS AN *entity*.

Thus defined, an ontology is a very powerful tool for constraining the form of the different entities that may be part of the sort system, since it rigidly defines a finite set of sorts with a fixed internal structure.
Given this, it becomes very important to draw a clear distinction between a sort, understood as a partial description of a linguistic object, and the object itself, since descriptions are always partial, whereas linguistic objects (representations) are always complete. In other words, the sort system defines a set of completely specified representations. I presume that the reader has already established the connection between the sort system and a grammar in the traditional sense, and between the collections of objects defined by the sort system and the infinite set of representations defined by a grammar (remember: the goals of the processing algorithm). Nothing new here apart from the fact that we have sensibly enriched the structure and the organization of the grammar. In fact, we are also in a position to clarify the relation that exists between the declarations in the grammar and the objects in the set of representations: between a grammar, expressed in some formal language, and the collection of representations it defines exists a semantic relation. That is, every declaration in the grammar describes (i.e., denotes) a set of representations that is a proper subset of the whole set of representations; the more specific is a declaration, the smaller will be its denotation, and conversely.[13]

Thus, any of the following declarations, even if partial, is a possible description. Notice, moreover, that the intersection of each of the respective denotation sets is not empty:

[12] In this paper, I will use interchangeably the terms 'ontology' and 'sort system', as well as 'type' and 'sort', although it is not entirely obvious that the latter mean exactly the same thing. The content of what follows is partly based on the works of Carpenter (1992) and King (1994).

[13] Thus, the set of representations is roughly equivalent to a Herbrand Universe; see Pereira & Shieber 1987: 88.

a.

$$\left[\begin{array}{l} \textit{sign} \\ \text{CAT } \textit{verb} \\ \text{FLEX } [1]\left[\begin{array}{l}\textit{infl}\\ \text{NUM } \textit{sing}\\ \text{PERS } 3a\end{array}\right] \\ \text{ARGS } \left[\begin{array}{l} \textit{args} \\ \text{SUB } \left[\begin{array}{l}\textit{sign}\\ \text{CAT } \textit{noun}\\ \text{INFL } [1]\\ \text{CASE } \textit{nom}\end{array}\right] \\ \text{OBJ } \left[\begin{array}{l}\textit{sign}\\ \text{CAT } \textit{noun}\\ \text{CASE } \textit{acc}\end{array}\right]\end{array}\right]\end{array}\right]$$

b.

$$\left[\begin{array}{l} \textit{sign} \\ \text{CAT } \textit{verb} \\ \text{ARGS } \left[\begin{array}{l} \textit{args} \\ \text{SUB } \left[\begin{array}{l}\textit{sign}\\ \text{CAT } \textit{noun}\\ \text{INFL } \textit{infl}\\ \text{CASE } \textit{nom}\end{array}\right] \\ \text{OBJ } \left[\begin{array}{l}\textit{sign}\\ \text{CAT } \textit{noun}\\ \text{CASE } \textit{acc}\end{array}\right]\end{array}\right]\end{array}\right]$$

c.

$$\left[\begin{array}{l} \textit{sign} \\ \text{CAT } \textit{verb} \\ \text{INFL } [1]\left[\begin{array}{l}\textit{infl}\\ \text{NUM } \textit{sing}\\ \text{PERS } 3a\end{array}\right] \\ \text{ARGS } \left[\begin{array}{l} \textit{args} \\ \text{SUB } \left[\begin{array}{l}\textit{sign}\\ \text{CAT } \textit{noun}\\ \text{FLEX } [1]\\ \text{CASE } \textit{nom}\end{array}\right]\end{array}\right]\end{array}\right]$$

d.

$$\left[\begin{array}{l} \textit{sign} \\ \text{CAT } \textit{verb} \\ \text{INFL } \left[\begin{array}{l}\textit{flex}\\ \text{NUM } \textit{sing}\\ \text{PERS } 3a\end{array}\right]\end{array}\right]$$

The descriptions above give us the opportunity to appreciate some of the properties of the user language that is standard in most unification-based formalisms. First, I keep the convention of using small caps for attribute names and italics for sort names; square brackets are used to indicate the different levels of structure. As may be seen, atomic sorts are written directly as attribute values. When a value is complex, its sort is written at the top-left corner of the bracket. Indexes, expressed as digits enclosed in brackets, indicate structure sharing. Thus, for example, in (a) the INFL attribute of the verb and the INFL attribute of the subject have a token identical value. Let us turn now to some relations that exist among the different descriptions.

Given two partial descriptions A and B, if the intersection of their respective denotation sets is not empty, we can say that some relation exists between them, since the information that each description contains is not incompatible with the information in the other. When this happens, we may state that there is a third description C, more specific that any of the previous ones, that contains exactly the same information as A and B. We say that C is the *unification* of A and B. For example, (g) below is the unification of (e) and (f):

e.

$$\begin{bmatrix} \text{CAT } \textit{noun} \\ \text{INFL } \begin{bmatrix} \text{NUM } \textit{sing} \end{bmatrix} \end{bmatrix}$$

f.

$$\begin{bmatrix} \text{CAT } \textit{noun} \\ \text{INFL } \begin{bmatrix} \text{PERS } 3\textit{a} \\ \text{GEN } \textit{masc} \end{bmatrix} \end{bmatrix}$$

g.

$$\begin{bmatrix} \text{CAT } \textit{noun} \\ \text{INFL } \begin{bmatrix} \text{NUM } \textit{sing} \\ \text{PERS } 3\textit{a} \\ \text{GEN } \textit{masc} \end{bmatrix} \end{bmatrix}$$

Notice that unification is an operation that closely resembles set union, with the peculiarity of being sensitive to the compatibility of the information contained within the descriptions. Thus, if the value of he cat attribute in (e) was *verb* instead of *noun*, its unification with (f) would not be defined.[14] Given this property, it is obvious that the unification of two descriptions is a new description containing at least as much information as the former, that is, (g) subsumes (e) and (f). Formally:

i. $A \wedge B = C$
ii. $C \leq A$
iii. $C \leq B$

If we assume that the set of all valid descriptions for some set of representations includes a minimal description, the less specific of all, subsumed by every other description (symbolized T), and a maximal description that subsumes all others (symbolized by ⊥), we can define the following properties of subsumption and unification:

[14]This is not completely true, as, for purely formal reasons, sometimes it is convenient to assume that the unification of two incompatible descriptions is defined, yielding a special result.

SUBSUMPTION	
i. Reflexive	$\forall A, A \leq A.$
ii. Transitive	$\textit{if } A \leq B \textit{ and } B \leq C, A \leq C.$
iii. Antisymmetric	$\textit{if } A \leq B \textit{ and } B \leq A, A = B.$
UNIFICATION	
i. Idempotent	$A \wedge A = A.$
ii. Commutative	$A \wedge B = B \wedge A.$
iii. Associative	$(A \wedge B) \wedge C = A \wedge (B \wedge C).$
iv. Identity	$A \wedge T = A.$
v. Zero	$A \wedge \bot = \bot.$

We are reaching the end of the paper, so let's summarize some important points I've tried to make and draw some conclusions. In the two final sections of the paper, I developed a conception of grammar in terms of a theory of representations. According to this idea, a grammar is a (hopefully finite) collection of declarations defining the (presumably infinite) set of possible representations. We've also seen that, formally, we can conceive of these representations as a special kind of graphs we called feature structures over which we can impose a number of formal and substantive constraints (e.g., that they are sorted, that sorts have appropriate attributes, and so on.). From this perspective, the representations in the set defined by the grammar must be understood as complete objects, standing in a semantic relation with the declarations of the grammar (i.e., a representation is always in the denotation set of some grammar declaration), whereas declarations are always partial descriptions of representations (of course, they may be more o less specific, but they are always partial). Moreover, the fact that declarations are partial has another interesting consequence: to the extent that two descriptions are informationally compatible, we can define an operation of unification to combine them into a larger, more specific description; this new description is said to subsume the other two, as it contains as much compatible information as any of them.

This very general idea of a grammar as described here is at the core of most feature and unification-based linguistic frameworks like HPSG (Pollard & Sag 1994, Sag & Wasow 1999) and it puts quite a lot of effort on one of the crucial prerequisites that, as I insisted in section 2, a linguistic theory must satisfy: developing a precise and well defined conception of the nature and form of representations. Recall that this is a very important point at the time of considering a problem from a computational perspective, as the form of representations may have critical consequences at the time of considering the problem of NLP.

As we saw in section 3, there exist powerful and efficient techniques for processing simple CFGs, but, as I noted, the theory of CFGs does not qualify as an adequate linguistic theory, since it is unable to capture some important facts about natural languages. This reason alone led me to propose a different way of looking at grammars and linguistic objects. In presenting this conception, I didn't go into the details of linguistic analysis, as this was not my main goal here—although a quick look to any of the references about HPSG, for example, will reveal that there's been considerable progress in this area, but rather to show how one may do a good work in linguistic theorizing without neglecting formalization; in fact, if my arguments in section 2 are true, one cannot neglect formalization. The point now is, if unification-based models provide an adequate framework for developing a theory of representations that is both linguistically and formally sound, how do they behave when considering processing issues? Recall that there is a trade-off between the theory of representations and the theory of processes, so this is legitimate question to ask. At the end of section 3, I pointed out that parsing algorithms for CFGs may be easily extended to process grammars which make an extensive use of features and unification (as unification is a 'natural' operation is some programming languages like Prolog, for example), but this is true as long as these grammars contain what we may call a 'context-free backbone', i.e., that features in representations are arranged along a typical phrase structure tree; this is, for example, the technique used in the PATR II formalism (Shieber et al. 1983; Gazdar & Mellish 1989, Ch. 7) where representations are standard phrase-structure trees with feature structures in their nodes instead of atomic symbols. The point is that we may not want to do that. As some recent research in unification-based phonology suggests (Aguilar, Balari & Marín 1998a,b), it may well be the case that we need more complex representations for which the classical 'context-free backbone' approach may be insufficient. In this case, a solution to this problem may be that processing reduces to a task of building a complex representation through the successive application of operations of unification (along the lines of what King (1994), in a different context, termed an 'elaboration'), a possibility that I've explored with some more detail in Balari (1999). Whatever the outcome of these speculations might be, however, I believe nothing of that sort could have been done without taking the computational perspective.

6 References

Aguilar, L., S. Balari and R. Marín. 1998. "Constraint-based Phonology: A case study of some Spanish phonological phenomena". Presented at the Workshop on Declarative Phonology, Centre for Cognitive Science, University of Edinburgh.

Aguilar, L., S. Balari and R. Marín. 1998. "Un análisis contrastivo de algunas diferencias fonológicas entre el catalán y el español". Presented at the XXI Congreso de la Sociedad Española de Lingüística, Madrid.

Balari, S. 1999. "Formalismos gramaticales de unificación y procesamiento basado en restricciones", in Gómez Guinovart, J., A. M. Lorenzo Suárez, J. Pérez Guerra and A. Alvarez Lugrís (eds.), *Panorama de la investigación en lingüística informática. Volumen monográfico de la Revista Española de Lingüística Aplicada*. Asociación Española de Lingüística Aplicada, Logroño, pp. 117-151.

Carpenter, B. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge.

Chomsky, N. 1970. "Remarks on Nominalization", in Jacobs, R. A. and P. S. Rosenbaum (eds.), *Readings in English Transformational Grammar*. Ginn and Co., Waltham, MA. Reprinted in Chomsky, N. 1972. *Studies on Semantics in Generative Grammar*. Mouton, The Hague, pp. 11-61.

Dennett, D. C. 1998/1985. "Can machines think?", in *Brainchildren. Essays on Designing Minds*. The MIT Press, Cambridge, MA, pp. 3-29.

Dertouzos, M. 1997. *What Will Be*. HarperCollins, New York, NY.

Frank, R. H. 1988. *Passions within Reason. The Strategic Role of the Emotions*. W. W. Norton, New York, NY.

Gazdar, G., E. Klein, G. K. Pullum and I. A. Sag. 1985. *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gazdar, G. and C. S. Mellish. 1989. *Natural Language Processing in PROLOG*. Addison Wesley, Reading, MA.

Hauser, M. D. 1996. *The Evolution of Communication*. The MIT Press, Cambridge, MA.

Jackendoff, R. 1987. *Consciousness and the Computational Mind*. The MIT Press, Cambridge, MA.

Jackendoff, R. 1997. *The Architecture of the Language Faculty*. The MIT Press, Cambridge, MA.

Kaplan, R. M. and J. Bresnan. 1982. "Lexical-Functional Grammar: A Formal System for Grammatical Representation", in Bresnan, J. (ed.), *The Mental Representation of Grammatical Relations*. The MIT Press Cambridge, MA, pp. 173-281.

Kay, M. 1980. *Algorithm Schemata and Data Structures in Syntactic Processing*. Report CSL-80-12, Xerox PARC.

King, J. P. 1994. “An expanded logical formalism for Head-driven Phrase Structure Grammar”. Seminar für Sprachwissenschaft, Eberhard-Karls-Universität, Tübingen, [Available on line at http://www.sfs.nphil.uni-tuebingen.de/~king/]

Langton, C. G., C. Taylor, J. D. Farmer and S. Rasmussen, eds. 1992. *Artificial Life II*. Addison-Wesley, Reading, MA.

Marr, D. 1982. *Vision*. Freeman, New York, NY.

Peacocke, C. 1986. “Explanation in computational psychology: Language, perception and Level 1.5”, *Mind & Language* **1**: 101-123.

Pereira, F. C. N. and S. M. Shieber. 1987. *Prolog and Natural Language Analysis*. CSLI Publications, Stanford, CA.

Pollard, C. J. and I. A. Sag. 1987. *Information-Based Syntax and Semantics 1: Fundamentals*. CSLI Publications, Stanford, CA.

Pollard, C. J. and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, IL.

Renfrew, C. and E. B. W. Zubrow, eds. 1994. *The Ancient Mind. Elements of Cognitive Archaeology*. Cambridge University Press, Cambridge.

Sag, I. A. and T. Wasow. 1999. *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.

Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson and M. Tyson. 1983. “The Formalism and Implementation of PATR-II”, in Grosz, B. J. and M. E. Stickel (eds.), *Research on Interactive Acquisition and use of Knowledge Final Report*. SRI International, Menlo Park, CA, pp. 39-79.

Uszkoreit, H. 1986. “Categorial Unification Grammars”, *Proceedings of the 11th International Conference on Computational Linguistics*, Bonn.

Wahlster, W. (ed.). 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, Berlin.

Wilks, Y. A. and K. Sparck Jones. 1983. “Introduction: A little light history”, in Sparck Jones, K. and Y. A. Wilks (eds.), *Automatic Natural Language Parsing*. Ellis Hornwood, Chichester, pp. 11-21.

Zeevat, H., E. Klein and J. Calder. 1987. “Unification Categorial Grammar”, in Haddock, N., E. Klein and G. Morril (eds.), *Categorial Grammar, Unification Grammar and Parsing. Edinburgh Working Papers in Cognitive Science, 1*. Centre for Cognitive Science, University of Edinburgh, Edinburgh, pp. 195-222.