Links & Letters 1, 1994 9-23

Computational linguistics: a brief introduction

Anna Espunya i Prat
Dpt. de Filologia Anglesa i Germanística
Universitat Autònoma de Barcelona

December 1992

ABSTRACT

Computational Linguistics is an interdisciplinary field of study that encompasses knowledge mainly from three areas: Linguistics, Computer Science and Logic. This article is an introduction to the analysis of language from a Computational Linguistics approach, to the first questions linguists face when they need to make natural language fit into structures that computers can manipulate. Easy examples illustrate current existing approaches to the morphology, syntax and meaning of language, presenting some of the difficulties and challenges faced in this field. A list of basic introductory readings is provided in the last section.

1. Introduction: computers and human language

For many of us it would be difficult to think of a world without computers. They are part of our existence at all scales: powerful computers in charge of the security and finances of countries, controlling airports, dams, hospitals and city traffic; domestic computers that we use as word processors, home economists, or entertainers. And recently, even as teachers.

As they become more popular, companies make them more and more friendly, easier to use. However, communication between us and them is not yet based on our language. Even if it is as simple as clicking the mouse on one command of the menu, much in our interaction with computers needs to be adapted to their possibilities.

Making computer systems that understand and speak any human language (also called natural language) is the main goal of Computational Linguistics, an interdisciplinary field of study that encompasses knowledge mainly from three areas: Linguistics, Computer Science and Logic. Computational Linguistics appeared in the late 1950s with the first work in Machine Translation, name given to the translation from one natural language into another performed by

the computer without the help of a human translator. Since those pioneering days there has been a steady increase in the quantity, variety and quality of the applications. Apart from machine translation, two of the most important lines of research and development have been man-machine interfaces and information retrieval systems:

Man-machine interfaces are programs that allow communication between the computer and its human user. For instance, if you are using a database, you may want to ask questions in English instead of using the command language provided by the database.

Information retrieval systems try to respond to the need of finding only the relevant information in large corpora of texts. Given that much of the information people use is still in a natural language form (e.g., books, journals, reports, etc.), every time a specific piece of information is needed, someone has to actually read all potential sources. Instead, we may want to ask a program to do the job for us. The system will read, select and retrieve from a natural language text the information we need.

This article is not intended as an introduction to concrete Computational Linguistics applications. Neither is it meant to be a critical review of the most recent developments in this field. And finally, it is not intended to contribute to the debate on the possibilities and limitations of computers as real natural language speakers or the more general debate on the philosophy of Artificial Intelligence. Those are topics which occupy the pages of specialized journals, books and presentations in conferences. This article is an introduction to the analysis of language from a Computational Linguistics approach, to the first questions linguists face when they need to formalize a language, to make it fit into structures that computers can manipulate. It is addressed to people with a background in language studies and some familiarity with personal computers, at the user level[1].

2. Symbols and languages[2]

Computers are very general symbol manipulation machines. Symbols are made up of zeroes and ones and they can represent numbers but also more complex objects like words, sentences, syntactic trees, etc. Take, for example, the word "IS". Many computers will represent this word as the sequence of two bytes[3] 10001001 and 10010011 corresponding to the letters I (137) and S (147) in the ASCII code (American Standard Code for Information Exchange). If you

1. As an entertaining introduction to the basic aspects of computers and an "antidote against anxiety produced by computers", I would recommend John Shore's *The Sachertorte Algorithm and Other Antidotes to Computer Anxiety*, 1985, Viking Penguin. There is Spanish translation published by Alianza Editorial (Alianza Universidad collection)
2. This section is based mainly on section 1.1 in the Introduction of Gazdar and Mellish (1989).
3. A byte is an information unit consisting of eight binary digits or bits.

give a computer a list of names and ask it to sort them in alphabetical order, "in" will precede "six" not because "i" precedes "s" in the alphabet, but because 137 is a smaller number than 147.

The point of the previous example is that computers represent linguistic objects in non-linguistic ways. We said before that computers are symbol manipulation machines. This means that, at the lowest level, computers perform very basic operations. Three decades ago, the instructions that programmers gave the computers (the programs) were very much machine oriented, which means that they had to tell the Computer to add numbers or move information around in the computer's memory. Programmers had to think of the problems in terms of numbers. Fortunately computer science has developed "high level" programming languages that allow to write more abstract instructions and concentrate on the problem which the program is intended to solve.

Among the high level programming languages used in Computational Linguistics projects, LISP and PROLOG are the most common[4]. Sometimes other languages are chosen, mainly to achieve higher speed in the execution of tasks, for instance "C" language and its variations.

3. Computational linguists and natural language

Before we start our tour of Computational Linguistics, we should make several observations on its goals compared to the goals of Theoretical Linguistics. The goal of Theoretical (generative) Linguists is to find the simplest theory of grammar that can account for our knowledge of language, and to explain the innate mechanisms that allow people to learn and use human language. They are concerned with language universals (principles of grammar which apply to all natural languages) and they focus mainly on grammatical competence (i.e., why people accept some sentences and reject others). Some particular linguistic theories also take as a fundamental goal to explain language use in ways which are psychologically plausible.

The goal of computational linguists is to write programs that can handle (understand or generate) as much natural language material[5] as possible. These programs are good but approximate solutions; they cannot deal with all sentences of a natural language, although they deal with the most common and interesting constructions. This fact, which is generally accepted by computa-

4. LISP and PROLOG are very different programming languages, the basic difference being in the underlying philosophy of how knowledge is stored and put to use and how problems need to be approached.

5. The term "Computational Linguistics" refers to studies on language independent of its physical realization as sound, which does not mean that computers cannot "hear" or "speak". Machine decoding of spoken material, i.e., hearing and identifying sentences as sequences of sounds is the goal of Speech Recognition, a field that combines work in Phonetics/ Phonology and Engineering. Machine production of spoken material ("uttering" a message) is the goal of Speech Synthesis, also an interdisciplinary field.

tional linguists, would be unacceptable for theoretical linguists, since it is part of their goal to account for all grammatical sentences of a language with their theory of grammar.

Work in Theoretical Linguistics is relevant for Computational Linguistics; all generalization efforts (e.g., by reducing a large variety of sentences to a small set of rules and constraints on those rules) made in the former to describe the grammar of a particular natural language, are essential for the latter, especially in the process of sentence analysis.

Work in Computational Linguistics is relevant for Theoretical Linguistics. The theory that underlies the more pragmatic work in Computational Linguistics views language understanding and generation as processes of symbol-manipulation in a rule-governed fashion. To the extent that Theoretical Linguistics is interested in all aspects of the language capacity (the abstract knowledge of language and how it is used), its work should be testable by systems which computational linguists design (Grishman 1986, Winograd 1983).

The task of constructing systems that understand or generate natural language is a complex one. It requires the integration of many kinds of data, linguistic (morpho-syntactic, semantic) and non-linguistic (knowledge of the domain of discourse). It also requires an effective use of all data. In this sense, designing and building a natural language application is an engineering task. One general strategy to make construction jobs easier is modularity, (i.e., dividing the problem into smaller subproblems. This notion is not foreign to linguistics. Traditionally, the language capacity is represented and studied as working in levels of structure: sounds, words, sentences. Linguists study the phonetics, the phonology, the morphology, the semantics, the syntax of a language and they assume the existence of levels or modules in human competence (e.g., Chomsky's Autonomous Syntax Principle). Even if the modular view is a simplification of the language capacity, it makes natural language systems flexible and easy to expand.

Just how much knowledge is used in the understanding or generation process depends on the purpose of the application. For many applications the essential task is analysing sentences, (i.e., determining what sentences mean. Some applications also require an analysis of suprasentential units, such as discourse and dialogue. In this presentation we will concentrate on sentence analysis.

4. Sentence analysis

We said before that the goal of sentence analysis is to determine what a sentence means. In computer terms, every input sentence has to be assigned a meaning representation of some kind. Two types of information are combined in a sentence that allow speakers to reach an interpretation of that sentence: its

syntactic structure and the lexical meaning of the words that compose it. Words usually take special shapes when they occur syntagmatically (i.e., in relation with the other words in the sentence). Thus the morphology of words is relevant for syntactic analysis. Once the structure of the sentence is determined, semantics is in charge of accounting for the way words and structures combine to express a meaning.

Sections 4.1 and 4.2 try to explain in very simple terms the tasks of the different modules of a sentence analysis system and the language facts such a system has to be ready to deal with, particularly those related to Morphology and Syntax. Section 4.3. introduces several basic issues related to Semantics.

4.1. Morphological analysis

Sentences are made up of words. Therefore, an important step in the process of syntactic analysis is looking up the words in the dictionary. The look-up stage is not a trivial one. When words appear in sentences they usually carry inflectional endings. Derivational affixes are also very common and they often change the part of speech of the root they attach to. And we should bear in mind that addition of affixes may involve spelling changes. Let me use some examples from English to illustrate the kind of problems a computational linguist faces when trying to formalize all morphological facts of a language[6].

Consider the conjugation of the English verb. Suppose the sentence we are trying to analyse is:

She always stops the joggers.

When looking up the word "stops", the result we expect is an analysis like:

{stop, verb} + {-s, 3rd person singular, present tense}

Our morphological analyser will have to divide the sequence "s t o p s" into two valid morphs, corresponding to the root {stop} and the inflectional ending {-s}, respectively. In order for the morphological analyser to identify the two morphs, they have to be in the dictionary which is consulted by the program. For now we may assume that both {stop, verb} and {-s, 3rd person singular, present tense} are in the dictionary. It does not seem important whether the items in the dictionary represent morphemes (abstract units) or morphs

6. In fact, English is a simple language as far as its inflectional morphology is concerned. Romance languages, just to mention a well-known family, display complex inflectional systems. Just as an example, consider the Spanish verb "acordar" (to agree). Some of its conjugated forms have root "acord-", with the stress on the inflection, e.g., "acordaremos" (1st. person plural, future tense), and other forms use the root "acuerd-", with the stress on the root, e.g. "acuerdas" (2nd person singular, present tense). The change "o" - "ue" is present in other verbs: "recordar", "volar", "sonar", "rogar", etc.

(particular realisations) because the representation of morphemes and the representation of morphs have exactly the same characters. Now let us suppose that the verb is not "stops" but "spies". In this case, the dictionary entries {spy, verb} and {-s, 3rd person singular} may never be identified by the analyser, which is dealing with the sequence "s p i e s". Probably the first solution that comes to mind is to have two entries for the verb "spy": {spy, verb} and {spie, verb}, with a tag signalling that they are the same verb, have the same meaning, etc. This may certainly be the only solution if the spelling alternation "y / ie" only affected the verb "spy": that is actually how morphological conditioning (e.g, *men* as the plural of *man*, *deer* as the plural of *deer)* is dealt with. However, since many words (nouns and verbs) show the same spelling change, it may be more efficient to incorporate the linguistic generalisation all roots ending in a consonant + *y* suffer a spelling change from *y* to *ie* before inflections that do not begin with *i*. Generalisations allow economy in the storage space, which in turn reduces the time required to find items.

A possible approach to the problem is to conceive the spelling change as a rule or an instruction to relate input words ending in *ie* in the appropriate conditions to dictionary items ending in *y* . Thus the dictionary only needs to include representations of morphemes (e.g., {spy, verb} but not of each concrete realization {spy, verb, non-third-person-singular}, {spie, verb, third-person-singular}. The morphology analysis program identifies the potential morphs in the input word. If a possible spelling change is detected (e.g., *ie* from *y* in the sequence "s p i e" in "s p i e s") , a small sub-program returns the candidate morpheme {spy}, which is then checked in the dictionary. This approach is called two-level morphology[7], because there are "surface" forms (those of input words) and "lexical" forms (those in the dictionary). The correspondence between the two levels is achieved through the small sub-programs.

4.2. Analysing syntax: parsers and grammars

In logical or mathematical terms, providing a syntactic analysis of a sentence is equivalent to finding out whether that sentence belongs to the set of possible sentences of a language, and if it is so, giving a representation of its structure, for instance in the form of a syntactic tree or a bracketed sentences with labels as subscripts. As we know, the number of sentences that are possible in a language is infinite. Fortunately, membership in this infinite set seems to be characterizable through a finite set of rules. Such characterizations of languages are called "grammars". Together with the lexicon (or dictionary), grammars can be considered the knowledge of language. In Computational Linguistics, the process of syntactic analysis is called "parsing".

7. The first implementations of "two-level" morphological analysers were written by Karttunen and his students for English (Karttunen, 1983) based on the work on Finnish by Kimmo Koskenniemi (see Barton et al. 1987 and Karttunen 1983).

What is required by a computer to parse a sentence of a particular language? First of all, the computer should have the knowledge of that particular language, (i.e., a grammar and a dictionary). Secondly, the computer should be able to use the knowledge of language. We need a program that takes the sentence as input and finds a combination of grammar rules and words of the dictionary that describe that sentence. Such programs are called "parsers".

Parsers make use of grammars. Both may be integrated in different ways, or rather, to various degrees. At one end, there are the parsers which contain instructions equivalent to grammar rules. The separation between grammar and parser is less clear. At the other end, there are systems which keep grammar and parser separated as different modules so that the same parser may be used with grammars of different languages. Another advantage of keeping the parser separated from the grammar is that it saves time and effort to the grammar writer (the person who writes rules that describe the particular language) and the programmer who builds the parser. If a grammatical sentence is not accepted by the parser, the problem could be in either component: the grammar might be wrong, or the parser might not work properly.

It is now time to set abstract discussions aside for a moment and see a description of a toy program for the syntactic analysis of language X, a subset of English. The program's components are a small grammar, an even smaller lexicon and the description of the parser. Even though we described morphological analysis, we will obviate this step here.

grammar of language X:

S → NP VP
NP → DET N
VP → Vt NP
VP → Vi

This grammar consists of a set of rewriting rules, rules that allow to build phrase structure trees of sentences. Each node has a symbol (syntactic category: S, NP, VP, etc. or word). The symbol on the left hand side of the rule is the mother of the symbols on the right hand side. This is called "immediate constituent approach", and it is just one of the possible approaches to grammar.

lexicon of language X:

the : DET
run : Vi (intransitive verb)
spy : Vt (transitive verb)
jogger: N
girl: N

Combining the rules and words of the toy grammar and lexicon (obviating the morphology), sentences like the following will be generated:

The girl spies the joggers.
The girls run.
The jogger runs.
The joggers spy the girls.
The jogger spies the girls.

Once we "know" language X (we have a grammar and a dictionary), we turn to a description in plain English of the sequence of actions performed by one particular parser.

The following is one of many possible parsing strategies. Since this is perhaps the most intuitive one, it will be used as example in this little parsing exercise. It is called bottom-up parsing: notice that we start building the tree from the words and the lower branches of the tree before we get to the top symbol. We will present simultaneously the instructions (in plain text) and the example parse (in italics).

Our example sentence is: "The girl runs."

1. Read word (from left to right). When all words in the sentence have been read, look in the tree-bag (where all potential branches found during the parsing process are stored) for a tree whose root is S. If there is one, display it to the user. If not, go to step 4.

1'. *"the"*

2. Look it up in the dictionary (this step includes morphological analysis).

- If it is there, find its grammatical category.

2'. *"the" is in the dictionary:* ***the : Det.*** *We have our first bit of the phrase structure tree:*

DET
The

3. a. Look in the grammar for a rule whose right hand side starts (from left to right) with the grammatical category of the word.

b. Do the following to all the symbols of the right hand side of the rule found in step 3 (a). When this step is finished, go back to step 1:

i. Read the next word, look it up in the dictionary, and find its grammatical category.

ii. If its grammatical category coincides with the following symbol on the right hand side of the rule, repeat step b.

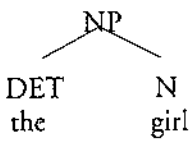
3'. a. *The rule* ***NP → Det N*** *has* ***Det*** *as its right hand side. We keep it as our current rule.*

b. current rule: NP → Det N

i. next word: girl . The dictionary has the entry girl: N. We have another potential branch of the tree.

N
girl

ii. N coincides with the following symbol of the right hand side of the rule after Det, i.e., NP → Det N . Since there are no more symbols left in the right hand side of the rule, we keep the rule NP → Det N in the tree-bag and we go back to step one. One of the potential branches is:



1. Next word : runs.

2.Dictionary entry: run : Vi. We have a new bit of the tree.

Vi
runs

3. a. rule VP → Vi.

b. The rule has no other symbols on its right hand side. We keep it in the tree-bag,

Vi
runs

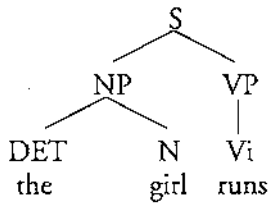
4. When all words in the sentence have been read, and rules have been found whose right hand sides had the grammatical categories of words, the tree-bag has separate rules that still have to be combined. The next action consists in finding a rule in the grammar whose right hand side symbols are left hand side symbols of the rules contained in the tree-bag. Rules in the grammar that match this specification are kept in the tree-bag.

The process finishes when a rule is found whose left hand side is the top node (Sentence). Then the syntactic tree is ready to be displayed.

4'. Relevant contents of our tree-bag:

NP → Det N
VP → Vi

Left hand sides: VP and *NP. Is there a rule in the grammar which looks like X → VP NP? The answer is no. However, there is a rule in the grammar that looks exactly like X → NP VP, namely S → NP VP. We keep it in the tree-bag and since its left-hand side is the start symbol S, a parse has been found for the sentence.*



We have oversimplified this illustration of a parsing process by our toy program for the sake of clarity. All computer programs have to consider all cases, and we skipped the negative ones and also those situations in which more than one rule fulfills the conditions. For instance, what would happen if our grammar did not contain a rule that combines a Determiner and a Noun into a Noun Phrase? What happens if there are two rules instead of one (e.g., NP → Det N, and NP → DET N AP), and the parser follows the wrong one first? Is there a way back from a wrong choice? All these questions and others related to them are taken into consideration when "real" parsers are implemented.

After parsing a sentence of language X, we should go back to English and the realization of the difficulties of parsing a natural language. Several facts deserve a comment.

1. The parser did not check agreement between the subject and the verb. Thus a sentence like "The girls runs" is possible in language X. In English there is agreement in person and number between subject and verb, in number between certain determiners and nouns, in person, number and gender between reflexive objects and their subjects, etc. Agreement checking is important in the parsing process and all systems have mechanisms to perform it.
2. The parser did not know whether all the obligatory arguments of the verb were in the sentence. Checking that the subcategorization requirements of the verb are met and that no extra arguments are added is a necessary step in parsing. Subcategorization frames are usually included in the lexical entries for verbs.
3. The grammar may have contained two rules, both equally valid for the parser at a given time. Suppose the grammar has the following rules:
   (a) NP → Det N-bar
   (b) NP → Det N-bar PP
   (c) N-bar → N PP

and suppose the parser has to provide an analysis for "The young man with the camera in the car". Once the parser has identified the PP "in the car", it has to choose between rules (b) and (c). In the first case, the young man is in the car. In the second, the young man may not be in the car, only the camera is.

Prepositional Phrase attachment is a case of global structural ambiguity: the parser correctly assigns two or more structures to a single sentence. Other sources of global structural ambiguity in English are coordination (e.g., "old men and ladies" gives two possible analyses) and noun-noun compounding. Parsing a globally unambiguous sentence may be affected by another type of structural ambiguity consisting in building up representations of constituents that ultimately do not lead to the final analysis. For example, when going through the sentence "The student whose advisor is a famous linguist read three books every week", the parser may decide that "a famous linguist read three books every week" is a sentence, a hypothesis that will have to be abandoned without any positive result. This time-wasting exploration is called local ambiguity.

4. The final observation is that the lexicon may have contained two words with the same form and different meanings, of the same or different grammatical category. In English many nouns can be used as verbs. A word form like "books" could be a 3rd person singular present tense verb form, or a plural noun. This is called lexical ambiguity. Often it gives rise to structural ambiguity, as in the typical example "I saw her duck", where the grammatical categories of "duck" involves two rather different structures for the sentence.

The reader might be thinking that human speakers are seldom mislead in our "parses" by ambiguity. Certainly, there are fewer cases, but we are not free from them. Try, for instance, to parse this: "The large rocks fell off the cliff", and now this: "The man rocks for three hours in his new concert". However, humans have one essential advantage: our dynamic knowledge of the world, of the people and circumstances around us, of what is plausible and what is not. Up to now, providing computers with a model of the world like ours has proved impossible. How to represent knowledge dynamically so that it can be used and updated by computers in problem solving (including natural language use) or prediction tasks is one of the central issues of Artificial Intelligence.

4.3. Meaning

Extracting the meaning of a sentence is not a straightforward task for a natural language system. Linguists and philosophers of language tend to distinguish those aspects of the meaning of an utterance that depend on its purpose and the context in which it is produced (pragmatic factors) and the semantic aspects of the sentence uttered, its truth conditions. This section sketches some of the

questions which need to be addressed in the process of extracting the semantic factors of the meaning of a sentence.

The first question, and the most important one is "what is (a) meaning ?", i.e., how should (a) meaning be *represented* in a computer (a symbol manipulation machine)? The answer is related to how meaning is going to be used. For example, an interface for a database requires input sentences to be interpreted as specific commands, since the user wants the database to perform concrete tasks. The set of possible meanings is the set of commands with all necessary specifications. A system that translates from English to Catalan may be designed to obtain some sort of representation of the meaning of the English sentence, so that the Catalan sentence can be produced from that meaning representation. The set of possible meanings is much larger and diverse than those required by the database interface.

Once we know how meaning is used in our system, the next step is asking ourselves how to obtain it. Extracting the meaning of a sentence is part of the analysis process. Therefore we need to consider whether the syntactic structure obtained in the parsing stage is relevant for the semantic interpretation stage. Most, if not all, approaches to natural language semantics are based on the assumption that there is a consistent relation between syntax and semantics. At the basis of such approaches lies a crucial principle attributed to the philosopher Frege, the principle of compositionality, informally phrased as: "The meaning of the whole is a function of the meanings of the parts". If we apply it to sentence interpretation, the meaning of a sentence S depends on the meanings of its subparts (NP and VP), whose meaning in turn depends on the meanings of their subparts, e.g. verb and complements for VP. The last elements of meaning are words and their morphemes.

The philosophical logician Richard Montague developed in his influential work a natural language semantics based on first order logic which made a strict application of the compositionality principle (Dowty et al. 1981) in a rule-by-rule fashion. His approach was truth-conditional, i.e., the meaning of a sentence is just its truth value (True or False). For instance, a syntactic rule that combined a NP and a VP to form a sentence was accompanied by a semantic rule that combined the meanings of the NP and VP to give the truth value of the sentence. Thus "John runs" is true if and only if [John] is an individual with the property [runs].

Several questions arise in a compositional approach to semantics, like the representation of word meaning (i.e., how do we represent the meaning of "run"?) or the fact that not everything in the meaning of a sentence is obtained compositionally. The concept is controversial. Nevertheless, it still underlies current approaches to natural language semantics.

Truth-conditional semantics and compositionality do not necessarily go together. Many applications need more than "True" as the meaning of a sentence. They need all information that the sentence conveys. In such cases,

meanings are represented in unambiguous artificial languages called Meaning Representation Languages (MRLs). Once the meaning of the natural language sentence is given an unambiguous representation in the MRL, it is ready to be used by any application that uses the same MRL. For instance, a railroad company may give information to its customers automatically just with one phone call. The computer program (application) would analyse the customer's request, find its meaning and translate it into a query for the timetable and destinations database in the MRL. The answer could then be translated back from the MRL into natural language and given to the customer in spoken form through a speech synthesizer.

5. Final remarks

The section devoted to parsing introduced the problem of ambiguity. Solutions to the problem have occupied the minds and the time of computational linguists. Lexical ambiguity (one word form with more than one sense) has traditionally been overcome by making dictionaries that are specific to the domain of the project. For instance, in a translation project of medical texts, some word senses may never be necessary. Having them in the lexicon gives rise to unnecessary ambiguity. A less restrictive approach is using information from collocations and selectional restrictions to distinguish word senses. Such an approach is also effective against structural ambiguity.

One well known source of structural ambiguity is PP attachment. In "cut the meat with the sharp knife" (example inspired by Gazdar and Mellish 1989), the PP "with a sharp knife" may be attached at the N-bar node (as a modifier of "meat") or, alternatively, to the VP node as an instrumental argument of the verb "cut" . Suppose (1) the lexical entry of "cut" included selectional restrictions information (i.e., that it may take an instrumental argument), (2) that "meat" does not have modifiers of the instrumental type and (3) that the entry of "knife" is categorised as an "instrument". By checking the selectional restrictions of words, the PP attachment to "meat" could already be discarded.

With the growing tendency to improve parsing by incorporating semantic information (from subcategorization frames that include thematic roles to selectional restrictions and collocations), efficient organization of the lexicon has become an important issue in Computational Linguistics.

Finally, to number but a few of the most challenging areas in Computational Linguistics, we should mention the treatment of time, tense and aspect; anaphora, ellipsis, coordination; speech acts, the intentions of the speaker; and, why not, metaphor, irony, and humor.

6. If you want to read more

Gazdar and Mellish (1989), *Natural Language Processing. An Introduction to Computational Linguisitics*, has an extensive bibliography on Natural Language Processing issues classified into areas, with comments on the difficulty and relevance of the works. Grishman, R. (1986) *Computational Linguistics. An Introduction*, is also a comprehensive introduction to the field, less programming oriented than Gazdar and Mellish.

Since the background of our readers is related to Linguistics more than Logic or Mathematics, they may be interested in the status of Syntax in Computational Linguistics. Winograd, (1983) *Language as a Cognitive Process,* is a classic introduction to the topic. The book contains exercises and describes techniques in detail.

Several linguistic theories that underlie many of the current natural language systems are based on the concept of unification. Three of them are Lexical Functional Grammar, Generalised Phrase Structure Grammar and Head-Driven Phrase Structure Grammar. A good introduction can be found in Sells, P. (1985) *Lectures on Contemporary Syntactic Theories.* Vol 3 of the CSLI Lectures on Contemporary Syntactic Theories. Volume No. 4 of the same series provides a good introduction to the mechanism of unification.

For those with a GB preference, there is parsing more or less faithfully based on the Principles and Parameters Theory. They can find a variety of articles and approaches in Abney, S. (ed.) (1988) *The MIT Parsing Volume.* Also in Berwick, Abney and Tenny (eds.) (1991) *Principle-Based Parsing: Computation and Psycholinguistics.* Dordrecht: Kluwer Academic Publishers.

On the use of logic to represent meaning compositionally: Dowty, Wall and Peters, (1981). *Introduction to Montague Semantics,* the first step for any Computational Linguistics student.

On the topic of knowledge representation there is a collection of articles by Brachman and Levesque (eds.), called *Readings in Knowledge Representation.* Gazdar and Mellish (1989) provide extensive references on issues in the semantics of natural languages (chapter 8) and on the different knowledge representation formalisms (chapters 1 and 9).

Temporal reasoning (tense and aspect) is covered by several articles in the journal *Computational Linguistics*, vol 14.

7. References

BARTON, E. et al. (1986). *Computational Complexity and NLP.* Cambridge: MIT Press.

DOWTY, D. et al. (1981). *Introduction to Montague Semantics.* Dordrecht: Reidel.

GAZDAR, G. and MELLISH, C. (1989). *Natural Language Processing in PROLOG, An Introduction to Computational Linguistics*. Wokingham, England: Addison-Wesley.

GRISHMAN, R. (1986). *Computational Linguistics. An Introduction*. Cambridge: Cambridge University Press.

KARTTUNEN, L. (1983). *KIMMO: A General Morphological Analyser*. Texas: Texas Linguistic Forum 22.

WINOGRAD, T. (1983). *Language as a Cognitive Process. Vol I: Syntax*. Reading, Ma: Addison-Wesley.