Foundations of Computational Linguistics: Man–Machine Communication in Natural Language

Roland Hausser

(Friedrich-Alexander-Universität Erlangen-Nürnberg)

Berlin: Springer, 1999, xii+534 pp;

hardbound, ISBN 3-540-66015-1, $54.00

Reviewed by

Alexander F. Gelbukh

National Polytechnic Institute, Mexico

As the subtitle of Hausser's book suggests, it is not exactly on what is usually supposed by computational linguistics. Instead, the author treats computational linguistics as a science of human-machine communication in real-world situations, with the ultimate goal of constructing cognitive, autonomous talking robots. The author notes that while most of the linguistic literature concentrates on the description of internal properties of language, surprisingly little effort is devoted to a functional theory of language that would model the process of communication in all its aspects. In this book, he presents his own theory, which he calls the SLIM theory of language, aimed at treating in a consistent and uniform way all aspects of communication, both linguistic and extralinguistic (nonverbal perception and action)—though the book mostly addresses linguistic issues. The description of all stages of language analysis—morphology, syntax, semantics, pragmatics, and even logical reasoning—is based on a single formalism called LA-grammar, and more specifically, on a subclass of LA-grammars that has linear complexity. The motto "everything is time-linear" runs through the text.

The book is organized in four parts. The first part, "Theory of Language", gives an outline of the SLIM theory. It begins with a general introduction to computational linguistics for novices that explains what a text is and why we should want to use computers to analyze it. Then the author passes to the idea of language functioning in real-world communication. One of the cornerstones of the SLIM theory is the idea of *internal* representation of meanings (which is the I in the acronym SLIM): the meaning of words used in communication exists only as a mental image inside the cognitive agent and does not exist in external reality. Since communication is aimed at changing the partner's internal cognitive structures (knowledge, tasks, etc.), understanding and modeling these structures is of crucial importance in modeling communication.

As a simple example, a toy robot is described that is capable of simple visual perception and mental representation of geometric objects. One can affect the mental status of the robot by presenting geometric images to it; for example, when seeing three connected lines, the robot matches the image against its repertoire of expected types (triangles, squares, circles) and decides that it resembles a triangle more than a square or circle; only then it can determine the parameters (coordinates, size, angles) of this specific instance of a triangle. Such *matching* is the second cornerstone of the theory (the M in the SLIM acronym): perceived images are not stored directly within the cognitive agent but instead are matched against expected patterns; if a suitable pattern is found, the perceived image is classified, with the necessary degree of detail, as an instance of the corresponding general concept (type of objects). Language perception—the words we hear—is no exception: instead of showing a triangle to the robot, one can describe such a triangle verbally, which results in the same effect—the robot constructs and stores a specific instance of the concept *triangle*. Thus, language is viewed as one of the means of affecting the hearer's cognitive state, or, for a speaker, as one of the means of action in the external world. This idea is elaborated in the last part of the book.

The second part, "Theory of Grammar", develops a universal computational formalism that is then applied to all language analysis and logical reasoning tasks throughout the rest of the book. The formalism, called LA-grammar (for *left associative*), is similar to good old augmented transition networks (ATNs). Like any generative grammar, it describes a language by means of the rules of an algorithm that reads the input string symbol by symbol and at some moment either accepts it as grammatical or rejects it as ungrammatical. The analysis algorithm maintains a record of some internal state—say, a tape with special symbols being written and erased, starting with an empty tape. After a symbol is read from the input, a rule is sought that allows it to be accepted given the current internal state (the whole contents of the tape). If no such rule exists, the string is rejected. If more than one rule is found, all alternatives are continued in parallel. Each rule provides an instruction for changing the current internal state (the contents of the tape); it also enables some subset of the rules and disables the others. Only enabled rules are used at each step of analysis.

The manner in which the rules decide whether or not the new symbol is compatible with the current state is not specified by the definition of the LA-formalism, the only requirement being that the corresponding Boolean function be recursive, i.e., computable in principle. The same holds for the procedure that changes the internal state. This gives great freedom in implementing parsers with "memory" to handle phenomena such as long distance dependencies and discontinuous constituents, but it raises the problem of development of formalisms for specifying these functions and procedures. Depending on what restrictions are placed on these Boolean functions, procedures, and the sets of rules that can be enabled simultaneously, subclasses of LA-grammars called C3 (less restricted), C2, and C1 are defined, with C1 having linear complexity. These subclasses are orthogonal to (that is, independent of) Chomsky's hierarchy of context-sensitive (CSG), context-free (CFG), and regular grammars: some CS-languages are C1-languages and thus can be parsed in linear time by a suitable C1 grammar.

The author argues for the hypothesis that all natural languages—even those with non-context-free phenomena (if they exist)—belong to the C1 class and thus have linear complexity, which is the author's main argument in favor of using his LA-grammars in language analysis and against using traditional phrase-structure (PS) grammars and formalisms mathematically equivalent to them (in which he includes, for example, HPSG).

The third part, "Morphology and Syntax", introduces the basic concepts of morphology and syntax and shows how to write C1-grammars that in a uniform manner build, element by element, morphs out of letters: $l + o + v$, wordforms out of morphs: $lov + es$, and sentences out of wordforms: $loves + Mary$. Such a *linear*, automaton-type order of processing of the elements is the third cornerstone of the theory (the L in SLIM): the author argues that we produce and perceive utterances letter by letter, word by word, and this order is to be directly modeled in a functional model of language and—because of the author's requirement of direct application of the rules by the parser—in the linguistic description. This is an idea that I find highly arguable, as the linguistic competence engaged in the processing is hardly of linear character. The final cornerstone is *surface compositionality* (the S in SLIM): no zero elements are allowed.

As to morphology, such surface-compositional, linear processing leads to rejection of the ideas underlying, say, the KIMMO model that builds a graphical representation of the word on the fly on the basis of interdependencies between its parts: $lady + es = ladies$, $put + ed = put$. Instead, all possible graphical variants of a morph—allomorphs: $\{$*lady-*, *ladi-*$\}$, $\{$*-s*, *-es*$\}$—are predefined in the dictionary (or generated by a stand-alone algorithm), and compatibility rules decide what combinations are possible: $ladi + es$, $*lady + s$. Such a "morphist" approach to morphology is analogous to well-known lexicalist approaches to syntax. I believe this is a good (even if not very new) idea.

As to syntax, the same LA-grammar mechanism is used. Thus, the very notion of a tree-like syntactic structure is absent from the theory (LA-syntactic structure is always linear, as per the L in SLIM). A sentence is either accepted by the automaton after it has read the final punctuation or it is rejected at some step; the semantic operations that augment the transition rules directly assemble the semantic representation of the sentence in the same linear order. At the end, consistency (grammaticality) of the sentence is checked for valency and agreement: all valencies should be filled and all agreement conditions satisfied. At any moment, the next word can prove to be incompatible with the already-read part of the sentence; then the analysis fails. Examples of small grammars, for both morphology and syntax, are given for English and German, including an example of the treatment of a supposedly non-context-free construction in German.

The author's main point here is that C1-grammar can be parsed in linear time with the suggested syntactic parser, while traditional CFGs have polynomial (almost cubic) complexity. How then is the ambiguity problem handled? For example, one of the most challenging problems of parsing leading to high complexity is ambiguity of prepositional phrase attachment: *Put the block in the box on the table*; what is on the table—*box, block,* or *put*? What Hausser suggests (page 236) is simply to leave the prepositional phrases unattached, it being the pragmatic module's job to incorporate them into the semantic network. As far as I understand, he would treat the example above roughly as it were *Put the block. This is in the box. This is on the table*, where it is the business of semantics, not syntax, to identify what *this* refers to. Unfortunately, I did not find any further explanation of how the pragmatic module would deal with such fragments. As to the syntactic parser, unattached phrases do not present any problems to it, since no syntactic structure at all is considered in the SLIM theory.

The last part, "Semantics and Pragmatics", further develops the idea of crucial role of pragmatics in natural language—a good idea that in my opinion is surprisingly underestimated by the computational linguistics mainstream. The difference is the following: *semantics* deals with the meaning that is stored in the dictionary entry for the word once and for all. *Pragmatics* deals with the meaning that the word has in a specific act of communication (occurring in a specific place, time, and circumstances). *Look, a mushroom!* says a traveler to his companion, having noted a rock formation with a flat wide top and thin base. To identify the object referred to, the hearer tries to find an object in their common view that most plausibly (or less implausibly) matches the standard dictionary definition of a mushroom; in this act of communication, the referent of the word *mushroom* proves to be a rock formation.

Distinguishing the dictionary meaning from the context meaning allows the author to give an elegant solution to the problem of vagueness—which is perhaps the most valuable (though not completely new) idea of the book. How is it that the word *mushroom* is so vague as to be applicable even to rock formations? To what else then is it applicable? Should its dictionary entry describe this continuum of meanings? Which of these meanings are more direct than others? Hausser's answer is this: neither the dictionary entry of the word *mushroom* nor its referring to the rock formation is vague; what is tensile is the matching of the dictionary definition against the least implausible candidate available in the specific communication situation. Clearly, there are other approaches to semantics, such as invariant definitions. Hausser's idea is similar to that of prototypes, which in its turn has received much criticism—see Wierzbicka (1989).

Thus, the final representation of the analyzed utterance proves to be pragmatic rather than semantic in its nature, according to the distinction made above—though functionally it corresponds to what is called semantic representation in the frameworks that do not make this fine distinction. It is quite similar to the familiar old semantic network (though the author carefully avoids this term). All knowledge that the cognitive agent has is represented as such a semantic network, whose nodes and relations are built by the agent during either parsing and interpretation of linguistic input, or by interpretation of otherwise-perceived images such as visual forms, or by logical inference—thinking.

The last chapter of the book describes logical inference implemented as a simple LA-grammar that can take two facts and produce as output a new fact—their logical combination such as *or* or *and*. Possibly the author has more to say about how logical inference is supposed to be done in SLIM theory but there was not enough space in the book to say it; however, in the way it is presented (pages 494–495), such uncontrolled, purposeless adding of trivial relations to the network does not seem to be a good substitute for classical inference methods; rather it looks like a bad illustration of the universal applicability of LA-grammars.

The same approach is described for text generation: "The most general form of navigation is the accidental, aimless motion of the focus point through the contents of word bank [i.e., semantic network—A.G.] analogous to *free association* in humans" (page 477). A simple LA-grammar is used for such "aimless" navigation through the network, verbalizing the nodes that are passed by. Though no real-world examples are given, I expect that the utterances generated would resemble a delirium rather than a reasonable reaction of the system; again, possibly the author has better ideas about how to make such generation more purposeful but did not have the space to explain them. The only thing explained is how the system can answer simple *yes–no* and *wh-* questions, interpreting them as patterns for search in the network. Surprisingly, in direct contradiction with the author's desiderata (page 181), the grammar used for parsing is not used for text generation. Instead, quite another grammar—actually another mechanism stuffed with *ad hoc* solutions and additions to the general LA-grammar formalism—is suggested for this purpose.

In general, in spite of its wide coverage, solid introduction, and quite a few good ideas, in its new proposals the book impressed me as yet another manual on building toy systems, especially in its treatment of semantics, reasoning, and text generation. It is good news and bad news: people who need to build a simple question-answering system or talking robot could find the suggested approach useful. On the other hand, the book does not provide any deep linguistic discussion, considering mostly the *John loves Mary* kind of artificial examples.

One of the main innovations introduced in the book is the LA-grammar formalism that—unlike traditional PS grammars—satisfies the author's eight desiderata for a generative grammar (page 180). Unfortunately, three very important requirements are absent from the author's list.

First, a grammar for a talking robot should be robust enough to understand incomplete or slightly ungrammatical sentences. The algorithm presented by the author, however, just rejects the sentence and aborts the analysis process when the next word read is unexpected. The author defines a generative grammar as a device "to formally decide for any arbitrary expression whether or not it is grammatically well-formed" (page 130) without any option of somehow processing (understanding) an input expression that it would not generate. Given this definition, it is strange that the author has chosen a *gen-*

erative grammar as the basis for a *functional* model of language. In communication, we do not worry much about whether or not the utterance we hear is grammatical but instead about what it means, and it is not a human-like behavior for a talking robot to fail to understand a whole sentence only because of a split infinitive or misplaced comma. Are there any alternatives to generative grammars that are more appropriate for talking robots? For example, Mel'čuk's Meaning ⇔ Text theory (Steele 1990) directly describes the translation of texts into meanings and vice versa.

Second, grammar formalisms should allow for the easy maintenance and extension of large grammars. The author argues that LA-grammars are easy to debug since the LA-parser executes the grammar rules directly, whereas traditional PS parsers translate the grammar rules into internal tables, which makes it difficult to track what actions of the parser corresponds to what rules. This is as true as the statement that Assembler programs are easier to debug than Prolog ones since at each moment you know exactly what line of your code is being executed. However, is it easier to *maintain* a large program in Assembler? An LA-grammar resembles the data structure used internally by the Earley algorithm: a list of all possible *continuations* in each possible state—with the exception that in this case you write this data structure manually. On pages 335–336, the author illustrates how easy it is to add a new rule to a toy four-line grammar. If this is considered easy, then I guess that a realistic-size LA-grammar would turn into a maintenance nightmare. Unfortunately (and probably not by accident) the author does not give any clear data on whether such realistic-size LA-grammars exist for any language, and if so, what the number of rules in such a grammar is and what its coverage of a real text corpus is.[1]

Third, grammar formalisms should directly support linguistic intuition, allowing

1 One of the maintenance problems with the LA-grammar as presented by the author is non-locality of changes: to add one rule, you have to adjust so-called rule packages in a bunch of other rules throughout the grammar, guessing what rules are to be adjusted and what not, which is very error-prone. I believe that the notion of a rule package (which is responsible for enabling some rules and disabling the others) in LA-formalism is mathematically redundant and methodologically harmful, or at least misused, though I do not have space here to discuss such technical details. Similarly, information on a word is scattered among the lexicon, rules, so-called variable definitions, and rule packages.

the linguist to write down his or her ideas more or less directly. The good news about LA-grammar is that it is based on notions well-known in general linguistics, valency and agreement, while the basic idea underlying PS grammars takes these phenomena as rather marginal (taking them seriously required the drastic changes that resulted in the emergence of HPSG). Actually, Hausser's syntax has a lot in common with the dependency approach (which he does not even mention), and this is the reason for its applicability to free-word-order languages. Is it then the long-awaited efficient computational formalism for dependency grammar? Possibly it is a good step towards such formalism. However, LA-grammars for natural languages presented in the book are rather counter-intuitive linguistically. While the notion of constituent is lost, the notions of dependency used in the book do not agree with linguistic tradition (Mel'čuk 1988). Often I found it hard to follow why a certain combination of rules happened to describe a certain type of sentence. Of course, this might be due to the way the author describes specific linguistic facts rather than to any inherent unsuitability of the formalism itself.

One of the author's main arguments in favor of his grammar and against PS grammars is that the latter have (almost) cubic complexity, whereas his C1-grammar is linear. I did not find this argument convincing at all from the engineering point of view. There are two important differences between an engineering linguistics and pure mathematics.

First, the length n of input sentences in real life is limited to, say, 1000 words (in the sense that the frequency of longer sentences decreases so fast that they will not affect the average time under any complexity). With this, it is not enough to say that C1-grammar has complexity an while PS-grammar has bn^3; the exact values of a and b do matter. The author does not mention the value of b for the PS-grammars, but he shows (page 211) that a is bounded by 2^R where R is the number of rules. Since in a realistic-size grammar, R is likely to be of the order of 1000, the argument about the advantage of the complexity an does not sound well-supported. Even knowledge of the coefficient b would not help a lot, as it is not clear what number of PS-rules would correspond (in what sense?) to a certain number R of LA-rules.

Second, the complexity of a grammar class is measured by the *worst case*: a grammar class has a complexity x if there exists some grammar in this class such that there exists an infinite series of long-enough sentences that parse in time x by this grammar. However, what matters in engineering practice is the *average* case for a *specific* grammar. Specific, since a specific grammar belonging to a high complexity class may well prove to parse much faster than the worse grammar of its class, even with the general algorithm, if the possible time-consuming behavior of the algorithm never happens for this grammar. Average, since it can happen that the grammar does admit hard-to-parse sentences that are not used (or at least not frequently used) in the real corpus. For example, Radzinsky (1991) proves that Chinese numerals such as *wu zhao zhao zhao zhao zhao wu zhao zhao zhao zhao wu zhao zhao zhao wu zhao zhao wu zhao*, for the number

$$500000000000000000500000000000000500000000000050000000005000,$$

are not context-free, which implies that Chinese is not a context-free language and thus might parse in exponential worst-case time. Do such arguments—no doubt important for mathematical linguistics—have any direct consequences for an engineering linguistics? Even if a Chinese grammar includes a non–context-free rule for parsing such numerals, how frequently will it be activated? Does it imply impossibility of processing real Chinese texts in reasonable time? Clearly, the average time for a specific grammar cannot be calculated in such a mathematically elegant way as the worst-case complexity of a grammar class; for the time being, the only practical way to compare the complexity of natural language processing formalisms is the hard one—building real-world systems and comparing their efficiency and coverage.

The above discussion raises the following questions. Since LA-grammar is similar to the Earley algorithm, can a linear-time C1-grammar be automatically built as the parser's internal representation for some new higher-level formalism that is linguistically more intuitive than the LA one—possibly resembling something like CSG, LFG, or HPSG? More specifically, can a subclass of CFGs, CSGs, or HPSG-like grammars be specified that allows efficient automatic translation into LA-grammar? Into C1-grammar? Can the corresponding converter be written that would give clear error messages if the grammar does not belong to this class?[2] Then: To what degree and in what form can the mathematical theory of complexity provide an estimation of the efficiency of parsing algorithms that is useful for engineering practice? More specifically, how can the average-case (rather than worst-case) complexity of a specific grammar (rather than a grammar class) be estimated? Can a program be written that at least in some cases verifies that the complexity of a specific CSG is less than exponential? How can non-equivalent grammar formalisms be compared as to their complexity, taking into account the practical restrictions on the length of the sentence? Finally: Can LA-formalism be used, in a linguistically intuitive and maintainable form, as a computational formalism for dependency grammars? Or, can the ideas of LA-grammar be useful in development of such a formalism?

The book is a compilation of the author's works mainly from 1973 to 1989. Incorporating into his framework ideas currently dominating the development of syntactic grammars, such as unification and hierarchical lexicon, would probably significantly improve the linguistic descriptions written in LA-formalisms. Such incorporation is possible without changing the definition of LA-grammar, since this definition does not pose any restrictions on the nature of the categories and operations used.

This book will be probably useful mostly to the developers of simple man–machine communication systems, as well as providing some useful ideas on implementation of parsers or giving the basis for the development of a new, possibly dependency-based, syntactic formalism.

2 An example of such compilation is the YACC language: it converts some (not all) CFGs into a regular grammar. If the grammar cannot be converted, the compiler points to the specific rule pair that is incompatible within the grammar.

Mel'čuk, Igor A. 1988. *Dependency Syntax: Theory and Practice.* State University of New York Press, Albany, NY.

Radzinsky, Daniel. 1991. Chinese Number-Names, Tree Adjoining Languages, and Mild Context-Sensitivity. *Computational Linguistics*, 17(3):277–299.

Steele, James (ed.). 1990. *Meaning-Text Theory: Linguistics, lexicography, and implications.* University of Ottawa Press.

Wierzbicka, Anna. 1989. Prototypes save: On the uses and abuses of the notion of 'prototype' in linguistics and related fields. Pages 347–366 in Tsohatzidis, Savas L. (ed.), *Meanings and Prototypes: Studies in Linguistic Categorization.* Routledge, London.

Alexander Gelbukh is the head of the Laboratory of Natural Language and Text Processing, a Professor and Researcher of the Center for Computing Research (CIC) of National Polytechnic Institute (IPN), Mexico City, Mexico. His current scientific interests include syntax of languages with free word order (such as Spanish and Russian) and applications involving semantic representations. He received his M.Sc. in mathematics from the Moscow "Lomonossov" State University, Russia, and Ph.D. in computer science from the All-Russian Institute of Scientific and Technical Information, Moscow, Russia, with a thesis on computational morphology of inflective languages. His address is: Centro de Investigación en Computación (CIC), IPN, Av. Juan Dios Bátiz s/n esq. Mendizabal, U.P. Adolfo López Mateos, Col. Zacatenco, CP. 07738, D.F., Mexico; email: gelbukh@cic.ipn.mx, gelbukh@earthling.net. The reviewer thanks Prof. Roussanka Loukanova for valuable discussion.