Theory-driven and Corpus-driven Computational Linguistics and the Use of Corpora

Stefanie Dipper, Mannheim

Computational linguistics and corpus linguistics are closely-related disciplines: they both exploit electronic corpora, extract various kinds of linguistic information from them, and make use of the same methods to acquire this information. Moreover, both were heavily affected by "paradigm shifts" from the prevailing empiricism of the 1950s, to rationalism, then back again with a revival of empirical methods in the 1990s.

Computational linguistics deals with the *formal modeling* of natural language. The formal models can be used to draw conclusions about the structure and functioning of the human language system. They also form the basis of implemented systems for the analysis and generation of spoken or written language, in a variety of applications. The methods applied in building these models are of different kinds since, as a result of the above-mentioned paradigm changes, work in computational linguistics has taken two different paths. Both branches of computational linguistics aim to build models of natural language, but each exploits different techniques: the rationalist's branch focuses on *theory-driven, symbolic, nonstatistical* methods, whilst the empiricist's branch focuses on *corpus-driven* and *statistical* techniques. As we will see later however, the distinction between the branches is these days less clear, and the two fields seem to be coming together again as people successfully combine concepts and methods from each field.

Obviously, the corpus-driven branch of computational linguistics has a natural affinity to corpus linguistics, and a shared interest in corpus exploitation. As a consequence, many research topics can be attributed equally well to either computational linguistics or corpus linguistics; examples include part-of-speech tagging (see article 25), treebanking (article 17), semantic tagging (article 27), coreference resolution (article 28), to name just a few. At opposite extremes of computational and corpus linguistics, the ultimate goals of corpus exploitation do however diverge: certain domains of corpus-driven computational linguistics aim to build "optimal" models "no matter how", and the particular corpus features that find their way into such models are not seen as interesting *per se*; in contrast, corpus linguistics could be said to target exactly these features, the "ingredients" of the models.

The theory-driven branch of computational linguistics does not overlap very much with corpus linguistics (except for their common interest in linguistic issues in general), although corpora do play a (minor) role in theory-driven computational linguistics, as we will show. So, we could more accurately rephrase the introductory sentence as follows: "*Corpus-driven* computational linguistics and corpus linguistics are closely-related disciplines."

Another side branch of research goes back to the early days of computational linguistics and is closely tied to artificial intelligence. Traditionally, this field focuses on modeling human behavior, including human actions like communicating and reasoning. A lot of research has gone into the formal description of world knowledge and inference drawing. These topics are nowadays seeing a revival, in the form of ontologies to encode concepts and the relations between them, and instances of the concepts. Current research on dialogue, such as human-machine communication, also draws heavily on this branch of computational linguistics. We will come back to the issue of world knowledge in the concluding section.

This article gives a survey of the research interests and concerns that are found in the theory-driven and corpus-driven branches of computational linguistics, and addresses their relation to corpora and corpus linguistics. Section 1 deals with theory-driven computational linguistics and Section 2 with the corpus-driven branch. In Section 3, we sketch the history of computational linguistics and trace the development of automatic part-of-speech taggers; this nicely illustrates the role that corpora have played and still play in computational linguistics. Section 4 concludes the article. Needless to say, this paper cannot do justice to all the work that has been done in computational linguistics. We hope however that the topics we address convey some of the main ideas and interests that drive research in this area.

1. Theory-driven computational linguistics

As a child of the paradigm shift towards rationalism, this branch of computational linguistics relies on the intellect and on deductive methods in building formal language models. That is, research is driven by *theoretical* concerns rather than empirical ones. The research issues addressed here often take up topics from theoretical linguistics. For instance, various syntactic formalisms have been the object of research in computational linguistics, such as Dependency Grammar (Tesnière 1959), HPSG (Head-Driven Phrase Structure Grammar, Pollard/Sag

1994), LFG (Lexical Functional Grammar, Bresnan 1982) or the Minimalist Program (Chomsky 1993).

Why is computational linguistics interested in linguistic theories? We see two main concerns of such research: firstly, the search for a complete, rigid and sound *formalization* of theoretical frameworks; secondly, concern for *implementation* of linguistic theories. We address both issues in the following sections.

1.1. The formalization of theoretical frameworks

As already stated, computational linguistics aims at a complete and sound formalization of theoretical frameworks. For instance, for the above-mentioned syntactic formalisms, computational linguists have defined formalisms that are mathematically well-understood: Kaplan/Bresnan (1982) for LFG, Kasper/Rounds (1986), King (1989, 1994) and Carpenter (1992) for HPSG, and Stabler (1997) with the "Minimalist Grammar" for the Minimalist Program. (Dependency-based systems come in a variety of realizations, and are in general formalized to a lesser degree than other theories.)

Other frameworks have started out as well-defined, purely-mathematical formalisms which were first studied for their mathematical properties, and have only later been exploited as the representational formats of linguistic theories. Such formalisms include TAG (Tree-Adjoining Grammar, Joshi/Levy/Takahashi 1975, Joshi 1985), CG (Categorial Grammar, Ajdukiewicz 1935, Bar-Hillel 1953), and especially its extension CCG (Combinatory Categorial Grammar, Ades/Steedman 1982, Steedman 1996); the linguistic relevance of these formalisms has been addressed, e.g., by Kroch/Joshi (1985) for TAG, and by Steedman (1985) for CCG.

What do these formalized theories offer? Armed with such a theory, computational linguists can explore the formal properties of the framework, such as its *structural complexity*. A commonly-used way of characterizing the complexity of a framework is by the form of its rules: For instance, a simple grammar rule like $N \rightarrow dog$ replaces (expands) a noun by the word *dog*, regardless of the noun's context. A more complex rule would be $N \rightarrow dog \,/\, DET \_$, which restricts the replacement to those contexts in which the noun is preceded by a determiner. Grammars are classified according to the most complex rule type that they contain: a grammar with rules like the second example above would be a member of the class of "context-sensitive" grammars. (The term "grammar" is often used to refer to syntactic rule systems. We call a grammar any linguistic rule system, including phonological, morphological, semantic, and pragmatic rule systems.)

This way of characterizing grammars has been introduced by Chomsky (1956, 1959). For each class of grammars, there is a corresponding class of languages that are generated by these grammars, and a corresponding abstract model, the "automaton", which represents an alternative way of defining the same class of languages. The first two columns of Table 1 display the four complexity classes as defined by Chomsky, with the most complex class at the top. Each class properly contains the simpler classes below it. This means, e.g., that for any context-free language (or "Type-2" language) we can define a context-sensitive grammar ("Type-1" grammar) to generate that language, but not vice versa. The resulting hierarchy of grammars and languages is known as the *Chomsky Hierarchy*. In the following paragraphs, we show how each of the above-mentioned linguistic frameworks relates to the Chomsky Hierarchy, then address issues of computational complexity (see last column of Table 1).

Structural Complexity		*Computational Complexity*
Grammar/Language Class	*Automaton*	
Type 0	Turing machine	undecidable
Type 1, context-sensitive	linear-bounded automaton	NP-complete
Type 2, context-free	pushdown automaton	$O(n^3)$
Type 3, regular	finite-state automaton	$O(n)$

Table 1: Structural complexity (Chomsky Hierarchy) and computational complexity

Unification-based formalisms, such as HPSG and LFG, are in general equivalent to a Turing machine (which generates Type-0 languages). The formalisms of TAG and CCG are less complex, but they can still express the famous cross-serial (= non context-free) dependencies observed in Dutch, Swiss-German, or Bambara (see, e.g., Savitch et al. 1987). TAG and CCG are appealing formalisms because they are only slightly more powerful than context-free grammars; that is, they do not use the full power of context-sensitive grammars and are therefore easier to compute than context-sensitive grammars in general. The complexity class of TAG and CCG is not part of the original Chomsky Hierarchy but lies between Types 1 and 2. Joshi (1985), who defined this class, coined the term "mildly context-sensitive".

The class of languages generated by finite-state automata or regular expressions (Type-3 languages) has received much attention since the early days

of research on formal languages (Kleene 1956; Chomsky/Miller 1958; Rabin/Scott 1959). In the following years, finite-state techniques became especially important in the domain of phonology and morphology: with SPE ("The Sound Pattern of English"), Chomsky/Halle (1968) introduced a formalism to express phonological processes, such as Place Assimilation (such as, "'n' in front of 'p' becomes 'm'"). The formalism defined an ordered set of rewriting rules which operated on phonological features such as [+/-nasal] and superficially resembled the rules of context-sensitive grammars: $\alpha \rightarrow \beta$ / γ _ δ ("replace α by β in context γ ... δ"). It turned out though, that the formalism, as used by the phonologists, was in fact equivalent in power to finite-state automata (Johnson 1972; Kaplan/Kay 1981, 1994). Kaplan and Kay showed this by means of a special type of finite-state automata, the so-called finite-state transducers. Their alphabet consists of complex symbols like 'n:m' (or feature bundles representing the phonemes), which can be interpreted as the "deep" (= lexical) and "surface" representations of phonological elements: phonemic 'n' becomes orthographic 'm'. In the formalism of Kaplan and Kay, the rewriting rules are applied in a sequential order. Another way of formalizing the mapping from lexical to surface form was the formalism of "two-level morphology", proposed by Koskenniemi (1983). In this formalism, declarative rules express parallel constraints between the lexical and the surface form. This formalism is again equivalent in power to finite-state automata. As the name suggests, the formalism has been used to formalize morphological phenomena (which in part overlap with phonological phenomena, but also include morphotactics).

Obviously, structural complexity is an important factor for the implementation of a linguistic theory, and implementation is the second concern of theory-driven computational linguistics. Theories that allow for more complex structures require more powerful programs to handle these structures than simpler theories do. For instance, a program that interprets context-sensitive rules (such as $N \rightarrow dog$ / DET _) needs some mechanism to look at the context of the node that is to be expanded, whereas programs for context-free rules can do without such a function.

Complexity is also seen as an issue for theoretical and psycholinguistics, since it might be related to questions of learnability and processability of language. A sample research question is: what bearing does a certain linguistic constraint have on the system's overall complexity? To answer such questions, computational linguists investigate the effects of adding, removing, or altering constraints, for instance, by (slightly) re-defining one of the "island conditions" or "move-alpha" in Minimalist Grammar. Does this result in a system that is more or less or equally

complex as the original system? (One might think, naively, that adding a constraint such as the "shortest move condition" would result in a more restrictive grammar, because it does not allow for many of the things that another system does allow; research has however shown that intuitions can be misleading.)

Another interesting research topic is the *computational complexity* (or parsing complexity) of a framework: given an input string of length *n* (e.g., *n* words or characters), how long does it take (at least) to compute an analysis, and how much storage space does the computation need? As one might expect, computational complexity parallels structural complexity: the simpler a grammar/language, the less time or storage space the computation needs.

For instance, from a computational point of view, finite-state automata (or regular/Type-3 grammars) are highly attractive, since there are efficient algorithms to process Type-3 languages which are linear in time. Thus, given an input of length *n*, these algorithms roughly need at most *n* steps to decide whether the input is accepted by a given finite-state automaton, i.e., to decide whether the input belongs to the language defined by that automaton. Using "big-O notation", we say that these algorithms run in O(*n*) time (see last column of Table 1). As a result, finite-state techniques have been and are used for a variety of tasks in computational linguistics, including speech, phonological and morphological processing, as well as syntactic analysis. Since, as is well-known, natural language syntax requires more powerful models than Type-3 grammars, the finite-state approaches *approximate* more powerful grammars, e.g., by a depth cut-off in rule application (and thus disallowing deeply-embedded structures).

For context-free grammars in general, there are also a number of relatively efficient algorithms, such as the Earley algorithm (Earley 1970) and the Cocke-Younger-Kasami (CYK) algorithm (Kasami 1965, Younger 1967), both of which run in O(n^3) time; that is, the algorithm roughly needs at most n^3 steps for processing an input string of length *n*.

Turning now to the class of Type-0 (Turing-equivalent) languages, Table 1 states that these are *undecidable*. This means that, even if provided with huge amounts of storage space and time, there is no general algorithm that would deliver an analysis for any arbitrary input (it could well deliver analyses (or rejections) for the vast majority of possible input data but not necessarily for all of them). The property of *decidability* pertains to questions such as: given a grammar and a sentence, is there a procedure that tells us whether the sentence is accepted/generated by the grammar, in other words, whether the sentence is

grammatical or not. The answer is that there is no such procedure for Type-0 languages in general.

As noted above, unification-based formalisms, such as HPSG and LFG, are in general equivalent to a Turing machine. This means that these formalisms would also be undecidable, in general. Since this is a highly problematic property, additional constraints have been proposed and added to the formalisms, to constrain their power and make them decidable. For instance, adding the "off-line parsability constraint" to the LFG formalism makes it decidable, in particular, "NP-complete" (Johnson 1988). As a result processing an LFG grammar on a *nondeterministic* Turing machine takes *polynomial* time ("NP" stands for "nondeterministic, polynomial"): $O(n^k)$, where k stands for some constant (which can be much larger than 3, as in the $O(n^3)$ time complexity of context-free algorithms). Computers themselves correspond to deterministic Turing machines however, so typical algorithms have to simulate non-determinacy and in this way actually take *exponential* time for LFG parsing ($O(k^n)$ -- here the input length n provides the exponens of the function rather than the basis; as a consequence, lengthening the input string has a drastic effect on computation time). Nonetheless, since natural languages are mostly equivalent to context-free languages, intelligent algorithms exploit this property and thus arrive at parsing in polynomial time, for most cases.

Abstract algorithms, such as the Earley algorithm, are used in mathematical proofs of complexity. The next step is to turn them into parsing algorithms, which determine mechanical ways of applying the grammar rules and constraints and using the lexicon entries so that, given an input string, the algorithm can finally come up with either an analysis (or multiple analyses) of the input string, or else with the answer that the input string is ungrammatical and no analysis can be assigned to it. This leads us to the second concern of theory-driven computational linguistics: implementing the formalized theories and parsing algorithms.

1.2. Implementation of the theoretical frameworks

Implementations of linguistic theories can be viewed as "proofs of concept": they prove that the formalizations are indeed sound and rigid, and exhibit the predicted complexity properties. An implementation consists of two parts: (i) a language-specific grammar (e.g. an LFG grammar for English) and (ii) a parser, which analyzes input strings according to that grammar (and the underlying formalism). It is the parser that knows how to "read" the grammar rules and to construct the trees or feature structures that constitute the analyses of the input strings.

The parsers are often embedded in "grammar development platforms", workbenches (software packages) which support the grammar writer in writing and debugging the grammar rules, e.g., by checking the rule format ("do all grammar rules end with a full stop?") or by displaying the output analyses in accessible formats. Important platforms for syntactic formalisms are: XLE (Xerox Linguistic Environment, from the NLTT group at PARC) for LFG implementations, LKB (Lexical Knowledge Builder, Copestake 2002) for HPSG grammars, but also used for implementing CCG grammars, and XTAG (Paroubek/Schabes/Joshi 1992) for TAG grammars.

For the implementation of phonological and morphological analyzers, widely-used tools are KIMMO (Karttunen 1983) and its free version, PC-KIMMO, from the Summer Institute of Linguistics (Antworth 1990), which embody the two-level rules of Koskenniemi (1983). The Xerox research groups have developed a collection of finite-state tools, which, among other things, implement rewriting rules (see, e.g. Beesley/Karttunen 2003). Computational linguists have also worked on formalizing and implementing semantics. CCG traditionally uses the lambda-calculus, building semantic structures in parallel with categorial structures (Steedman 2000). In the LFG world, the formalism of Glue Semantics has been both developed and implemented (Dalrymple 1999); in the HPSG world, MRS (Minimal Recursion Semantics, Copestake et al. 2005) has been applied.

An implementation does not only serve as proof of the sound formalization of a theoretical framework. It can also serve linguists by verifying their formalization of specific linguistic phenomena within this framework. Development platforms can support the linguist user in the formulation and verification of linguistic hypotheses: by implementing a grammar of, e.g., phonological or syntactic rules and lexicon entries, the user can verify the outcome of the rules and entries and experiment with variants. As early as 1968, Bobrow/Fraser implemented such a system, the "Phonological Rule Tester", which allowed the linguist user to define rewriting rules as presented in SPE, and to test the effect of the rules on data specified in form of bundles of phonemic features.

The earliest implementations consisted of grammar fragments or "toy grammars", which could handle a small set of selected phenomena, with a restricted vocabulary. With the advent of more powerful computers, both in speed and storage, and of the availability of large electronic corpora (see article 4), people started to work on broader coverage. Adding rules and lexicon entries to a grammar can have quite dramatic effects however, because of unexpected and, usually, unwanted interferences. Such interferences can lead to rules canceling each other out, or else they give rise to additional, superfluous analyses.

Interferences can provide important hints to the linguist and grammar writer, by pointing out that some grammar rules are not correctly constrained. The problem, though, is that there is no procedure to automatically diagnose all the interference problems of a new rule. A useful approximation of such a procedure is the use of *testsuites* (see article 17), which are representative collections of grammatical and ungrammatical sentences (or words, in the case of phonological or morphological implementations). After any grammar modification, the grammar is run on the testsuite, and the outcome is compared to previous test runs.

1.3. Theory-driven computational linguistics and corpora

We complete this section by briefly summarizing the main points of interest of theory-driven computational linguistics and then address the role of corpora and the relation to corpus linguistics. As the name suggests, computational linguistics deals with "computing linguistics": linguistic phenomena and theories are investigated with regard to formal correctness, and structural and computational complexity. A second aspect is the development and verification of language-specific grammars, in the form of implementations.

What role do corpora play in this field? Firstly, as in research in (corpus-based) linguistics, corpora serve in computational linguistics as a "source of inspiration"; they are used to obtain an overview of the data occurring in natural language and to determine the scope of the phenomenon that one wants to examine. Secondly, corpus data drive the usual cyclic process of theory construction: we start by selecting an initial set of examples that we consider relevant for our phenomenon; next, we come up with a first working model (in the form of a set of linguistic rules or an actual implementation), which accounts for our initial set of examples; we then add more data and test how well the first model fits the new data; if necessary, we adjust the model, such that it accounts for both the initial and new data; then we add further data again, and so on. Testsuites with test items (sentences or words) for all relevant phenomena can be used to ensure that the addition of rules for new phenomena does not corrupt the analysis of phenomena already covered by the model.

In the early days of (toy) implementations, *evaluation* did not play a prominent role. However with more and more systems being implemented, both assessment of the systems' quality (performance) and comparability to other systems became an issue. The performance of a system can be evaluated with respect to a standardized gold standard, e.g., in form of testsuites or corpora with annotations, such as "treebanks" (see article 17). Performance is usually

measured in terms of the grammar's coverage of the gold standard. Other measures include the time needed to parse the test corpus, or the average number of analyses. As we will see in the next section, thorough evaluation, according to standardized measures, has become an important topic in computational linguistics.

In the scenarios described above, both the analysis and the use of corpus data is mainly *qualitative*. That is, the data is inspected manually and analyses are constructed manually, by *interpreting* the facts and hand-crafting rules that fit the facts. Data selection and analysis are driven by theoretical assumptions rather than the data itself. In this respect, theory-driven computational linguistics is closely related to (introspective) theoretical linguistics -- and is unlike corpus linguistics.

An alternative strategy is to automatically derive and "learn" models from corpora, based on quantitative analyses of corpora. This method is more consistent with the empiricist paradigm, which relies on inductive methods to build models bottom-up from empirical data. The empiricist's branch of computational linguistics is addressed in the next section.

2. Corpus-driven computational linguistics

Up to the late 1980s, most grammars (i.e., phonological, morphological, syntactic, and semantic analyzers) consisted of knowledge-based expert systems, with carefully hand-crafted rules, as described in Section 1. At some point, though, manual grammar development seemed to have reached its limit and no further progress seemed possible. However the grammars had not yet arrived at a stage that would permit development of useful applications (something that was urgently requested by funding agencies). In general, common deficiencies of hand-crafted systems were:

(i) Hand-crafted systems do not easily *scale up*, i.e., they are not easily extensible to large-scale texts. As described in the previous sections, early implementations consisted of toy grammars, which covered a small set phenomena, with a restricted vocabulary. When such systems are augmented, e.g., to cover real texts rather than artificial examples, interferences occur that are not easy to eliminate. The grammars of natural languages are complex systems of rules, that are often interdependent and, thus, difficult to manage and maintain.

(ii) Hand-crafted systems are not *robust*. Real texts (or speech) usually contain many words that are unknown to the system, such as many proper nouns, foreign words, hapax legomena and spelling errors (or mispronunciations). Similarly, real texts contain a lot of "unusual" constructions, such as soccer results ("1:0") in sports news, verbless headers in newspaper articles, syntactically-awkward and semantically-opaque idiomatic expressions, and, of course, truly-ungrammatical sentences. For each of these "exceptions", some "workaround" has to be defined that can provide some output analysis for them. More generally speaking, "the system needs to be prepared for cases where the input data does not correspond to the expectations encoded in the grammar" (Stede 1992). In the case of spelling errors and ungrammatical sentences, it is obvious that workarounds such as additional rules or constraint relaxation risk spoiling the actual grammar itself and causing it to yield incorrect (or undesirable) analyses for correct sentences.

(iii) Hand-crafted systems cannot easily deal with *ambiguity*. Natural languages are full of ambiguities; famous examples are PP attachment alternatives ("The man saw the woman with the telescope") or the sentence "I saw her duck under the table", with (at least) three different readings. In fact, people are usually very good at picking out the reading which is correct in the current context, and indeed are rarely conscious of ambiguities and (all) potential readings. For example, Abney (1996) shows that the apparently impossible "word salad" sequence "The a are of I" actually has a perfectly grammatical (and sensible) NP reading, wich can be paraphrased as "The are called 'a', located in some place labeled 'I'" ('are' in the sense of 1/100 hectare). Ambiguity is a real challenge for automatic language processing, because *disambiguation* often needs to rely on contextual information and world knowledge. Moreover, there is a natural tradeoff between coverage/robustness and ambiguity: the more phenomena a grammar accounts for, the more analyses it provides for each input string. This means that after having arrived at a certain degree of coverage, research then has to focus on strategies of disambiguation.

(iv) Hand-crafted systems are not easily *portable* to another language. Development of a grammar for, e.g., Japanese, is of course easier for a grammar writer if she has already created a grammar for English, because of her experience, and the rules of "best practice" that she has developed in the first implementation. It is, however, not often feasible to reuse (parts of) a grammar for another language, especially if the two languages are typologically very different, such as English and Japanese.

For the initial purposes of theory-driven computational linguistics, these deficiencies were not so crucial. For *applied* computational linguistics, which

focuses on the development of real applications, the shortcomings posed serious problems. Thus researchers in applied computational linguistics sought alternative methods of creating systems, to overcome the deficiencies listed above. They found what they were looking for among the speech-processing community, who were working on automatic speech recognition (ASR) (and speech synthesis, TTS, text-to-speech systems). Speech recognition is often regarded as a topic of physical, acoustic engineering rather than linguistic research, and researchers had successfully applied statistical methods on a large scale in the late 1970s. With the success of the ASR systems, people tried, and successfully applied, the same methods in other tasks, starting with part-of-speech tagging, and then moving on to syntax parsing, etc. The large majority of today's applications in computational linguistics make use of quantitative, statistical information drawn from corpora.

Interestingly though, statistical techniques are not really new to the field of computational linguistics, which in fact started out as an application-oriented enterprise, using mainly empirical, statistical methods. The earliest statistical applications are machine translation (e.g., Kaplan 1950, Oswald 1952), speech recognition (Davis/Biddulph/Balashek 1952), optical character recognition (OCR, Bledsoe/Browning 1959), authorship attribution (Mosteller/Wallace 1964), or essay grading (Page 1967). It was only after the influential ALPAC report in 1966 (ALPAC 1966), and in the wake of Chomsky's work (e.g., Chomsky 1956, 1957, see article 3), that the focus of research switched to rationalism-based, non-statistical research (with the exception of speech recognition), and gave rise to the branch of research that we called "theory-driven computational linguistics" (this switch is addressed in more detail in Section 3). The severe shortcomings of the theory-driven (toy) implementations however led researchers to look for alternative methods, and to re-discover corpus-driven techniques. But it was not until the 1980s that people started to apply statistical methods (on a large scale) to tasks such as part-of-speech tagging, parsing, semantic analysis, lexicography, collocation or terminology extraction. Indeed it was even longer before statistical methods were once again reintroduced into research on machine translation.

The reasons mentioned so far for re-discovering statistical methods are rather pragmatic, technically-motivated reasons. Following Chomsky, one could say that corpus-driven approaches are misguided, since they model language *use*, i.e., performance rather than competence, in Chomsky's terms. However, many people today claim that competence grammar takes an (overly) narrow view of natural language, by restricting language to its algebraic properties, and that statistical approaches can provide insights about other linguistically-relevant properties and

phenomena of natural language, such as language change, language acquisition, and gradience phenomena (see, e.g., Klavans/Resnik 1996; Bod/Hay/Jannedy 2003; Manning/Schütze 1999, ch. 1). Thus there may be theory-driven reasons to re-focus on corpus-driven methods. In the next section, we present prominent concepts and methods used in statistical approaches and then discuss how statistical methods overcome the above-mentioned deficiencies.

2.1. Concepts and methods of statistical modeling of language

In Section 1, we introduced formal models of language in the form of formal grammars and automata as defined by the Chomsky Hierarchy. These models take words or sentences as input and produce linguistic analyses as output, e.g. part-of-speech tags or syntactic trees, or the input can be rejected as ungrammatical. *Statistical* (or *probabilistic*) formal models also take words or sentences as input and produce linguistic analyses for them. In addition, they assign *probabilities* to all input-output pairs. For instance, a statistical part-of-speech tagger might assign probability 0.7 to the input-output pair *book-NN* ("book" as singular noun) and 0.3 to the pair *book-VB* ("book" as verb base form). Ungrammatical and absurd analyses like *book-CD* (cardinal number) would receive very low or zero probability.

The models used in corpus-driven computational linguistics are probabilistic variants of the formal grammars and automata defined by the Chomsky Hierarchy. For instance, a *probabilistic context-free grammar* (PCFG) can be seen as an ordinary context-free (Type-2) grammar whose analyses are augmented by probabilities. Applying an implementation of a PCFG, e.g. for English, to an input sentence typically results in a huge number of different syntactic analyses (thousands or even millions), each supplied with a probability. The probabilities impose a natural order on the different analyses: the most probable analyses are the most plausible ones, and indeed most likely the correct ones. In sum then, the probabilistic models arrange the individual analyses of some input along a scale of probabilities, marking them as more or less plausible, relative to each other.

Due to their favorable computational properties, Type-3 models are again highly popular, just as in theory-driven computational linguistics. The most prominent probabilistic Type-3 models are *n-gram models* (which we describe in more detail below) and *Hidden Markov Models*. Most work on statistical syntactic parsers deals with PCFGs (Type-2 models), while comparatively less research has been devoted to probabilistic variants of Type-1 grammars.

Comparing statistical with non-statistical models, we could say that non-statistical models are simplified versions of statistical models in that they assign

just two "probability values": "1" to all grammatical inputs (= "accept"), and "0" to all ungrammatical ones (= "reject"). It is the task of the grammar writer to write the grammar rules in such a way that all and only the correct input words or sentences are accepted. In contrast, rules of statistical models are usually written in a very general way (similarly to underspecified rules in "universal grammar"), and may -- at least in theory -- even include absurd assignments such as *book-CD*, or rules that are ungrammatical in the language under consideration, such as *PP* $\rightarrow$ *NP P* in an English grammar. The probabilities that are assigned to these rules would be very low or equal to zero, and all analyses that these rules participate in would "inherit" some bit of the low probability and thus be marked as rather implausible.

Where do the probabilities for the rules come from? The models themselves only define the factors (*parameters*) that are assigned the probabilities. For PCFGs, these factors are context-free grammar rules. For simpler models, the factors can be word forms, part-of-speech (PoS) tags, sequences of PoS tags, pairs of word forms and PoS tags, etc. For instance, a model could define that the probability of a PoS assignment such as *book-NN* depends on the individual probabilities of the word and PoS tag in isolation ("How probable is it that the word 'book' occurs in some text, compared to all other words? How probable is the tag 'NN', compared to all other PoS tags?"). The question is then where the individual probabilities come from and how their probabilities are combined to produce the overall probability of *book-NN*.

The answer to the first question is that the probabilities can be derived from *corpus statistics*. The basic idea in statistical modeling of linguistic phenomena is to "take a body of English text (called a *corpus*) and learn the language by noting statistical regularities in that corpus" (Charniak 1993, 24). Put in technical terms then, most statistical modeling relies on the assumption that the *frequencies* of so-called events (the occurrences of a certain word, part-of-speech tag or syntactic configuration), as observed in a corpus, can be used to compute (or estimate) the *probabilities* of these linguistic events, and thereby to detect regularities and generalizations in the language.

For instance, an existing, non-statistical grammar can be augmented with probabilities by running the original grammar on a corpus and keeping a record of how often the individual rules are applied in the analyses (as in Black/Lafferty/Roukos 1992). Another way is to use annotated corpus data (e.g., treebanks) both to induce grammar rules, by reading off rules from the corpus, and to assign probabilities to them (e.g., Charniak 1996, Bod 1998). Finally, plain

text data, without annotations, can also be used to induce statistical grammars, such as syntactic parsers (van Zaanen 2000).

The process of calculating probabilities on the basis of a model and corpus data is called *parameter estimation* (or *training*): the fillers of the parameters are assigned probabilities, resulting in an instance of the formal model. A model instance can in turn be used to predict future occurrences of linguistic events, and thereby be applied to analyse previously-unseen language data. How the individual probabilities result from corpus frequencies and combine to an overall probability is a matter of probability theory. Statistical models are based on the (simplifying) assumption that their parameters are statistically independent. The individual probabilities of "competing candidates" therefore sum up to 1. In a PCFG, for instance, the probabilities of all rules with the same left-hand side (e.g., $NP \rightarrow N$, $NP \rightarrow DET\ N$, $NP \rightarrow ...$) sum up to 1. The probability of a PCFG analysis of a complete sentence is then computed as the product of the probabilities of the rules that participate in the analysis.

How the parameters are actually assigned their probabilities depends on the algorithm that is applied. For Type-3 models, a commonly-used estimator algorithm is the "forward-backward algorithm" (also called "Baum-Welch algorithm", Baum 1972); for PCFGs, the "inside-outside algorithm" is used (Baker 1979). Both types of algorithms are specific instances of a general algorithm called "Expectation Maximization" (EM, Dempster/Laird/Rubin 1977).

Having built a model instance, we need algorithms to apply the model to new input data, similar to the algorithms that interpret and apply the grammars within theory-driven computational linguistics; this task is often called *decoding*. The Earley and CYK algorithms for context-free grammars, which we mentioned in Section 1.1, can be modified and applied to PCFGs. For Type-3 and Type-2 models, the most prominent decoding algorithm is Viterbi (Viterbi 1967). As with theory-based computational linguistics, computational complexity of the frameworks is an important issue: the training and decoding algorithms mentioned above are of polynomial computational complexity.

We now address selected concepts and methods of statistical modeling in more detail.

2.1.1. Noisy channel

The task of decoding is neatly illustrated by the metaphor of the *noisy channel*. As already mentioned, many statistical approaches were inspired by work in the area of speech recognition. The aim of speech recognition is to map speech signals to words, and to this end, the acoustic signal must be analyzed. Very often the

system cannot uniquely determine a particular word, e.g., words such as "big" or "pig" are hard to distinguish on an acoustic basis only.

In his theory of communication ("Information Theory"), Shannon (1948) develops the metaphor of the noisy channel to describe problems such as these, which arise in the process of communication. According to the noisy-channel model, communication proceeds through a "channel" which adds "noise" to the signal: *original signal* → *noisy channel* → *perturbed signal*. For instance, a source word like "pig" might be "disturbed" by the channel and come out as or be misheard as "big". Or else a source word might be disturbed in that one of its letters is deleted, resulting in a spelling error. The aim of information theory, and cryptography in general, is to reconstruct (decode) the original source signal from the perturbed signal at the end of the channel. How can this be done automatically? For this, we combine three types of information: (i) the perturbed signal (which we have at hand); (ii) some measure of "similarity" between the perturbed signal and all source signal candidates (we prefer signal pairs that are rather similar to each other than unsimilar); (iii) the prior (unconditioned) probability of the source candidates, specifying how probable it is that the speaker actually uttered that source signal (we prefer source signals that are usual, frequent signals, over unfrequent ones).

For calculating similarity (ii) and prior probability (iii), we can use statistics derived from corpora: (iii) can be estimated on the base of a large (balanced) corpus, by counting word occurrences (or occurrences of parameters other than wordforms, as specified by the model at hand). We thus might learn that "big" is a highly probable word, and "pig" is less probable. For (ii), we need a list of word pairs that are commonly mixed up (or misspelt) and the numbers for how often that happens. This information can be calculated on the basis of a corpus that is annotated with the relevant information. In this way, we might learn that "pig" is often misheard as "big". Based on this information, we can calculate the optimal candidate for the source signal, which realizes the best combination of being similar to the perturbed signal and of being a probable word itself (although what counts as "similar" depends on the specific task).

2.1.2. Bayes' rule

The three components (i)-(iii) of the noisy-channel model stand in a certain relation to each other, and this relation is made explicit by an equation, called *Bayes*' *rule* (or *Bayes*' *law*). The equation captures the fact that we can swap the dependencies between source and perturbed signal: we are interested in determining the most probable source signal (what we are trying to reconstruct)

given a perturbed signal: $argmax_{source \in X}\ P(source|perturbed)$, where X is a set of alternative sources for the given "perturbed". According to Bayes' rule, this can be computed as $argmax_{source \in X}\ P(perturbed|source) * P(source)$, in which the arguments of the probability function P are switched. In fact we effectively applied this rule already in our informal reasoning of how to reconstruct the source signal: the first factor of the product expresses our vague notion of similarity (ii), the second factor captures the prior probability of the source signal (iii).

Bayes' rule is not restricted to linguistic applications but can be found in everyday reasoning. For instance, a doctor usually applies Bayes' rule in the diagnosis of a disease (example from Charniak 1993, 23). The doctor's task is to determine, given a certain symptom, the most probable disease ($P(disease|symptom)$). Often, the doctor does not know the values for all possible diseases (a symptom can co-occur with a number of different diseases). However, she knows quite well which symptoms are usually associated with a disease ($P(symptom|disease)$). Put differently, the doctor's knowledge is indexed by diseases, whereas the task at hand requires an index of symptoms. In addition however, the doctor knows how often a disease occurs ($P(disease)$), i.e., which diseases are worthy of consideration. With this knowledge, she can derive the most probable disease given the symptom ($P(disease|symptom)$).

2.1.3. N-gram models (= Markov models) and Hidden Markov Models

Up to now, we have applied the noisy-channel model to decode isolated words, and this approach seems sensible for simple tasks like spelling correction. For speech recognition and other tasks, however, it may make a big difference whether we look at isolated words or words in context: the prior probability of "big" might well be higher than that of "pig"; however in a context such as "The nice _ likes John", we know for sure that it is more probable that "pig" will occur than "big", and corpus data should somehow provide evidence for that. That is, we have in fact to deal with entire sentences (or even complete texts) rather than just words. We thus need to know the prior probability of source candidates that consist of entire sentences, and we need likewise to measure similarity between pairs of sentences. Obviously, however, we do not have corpora that contain all possible source sentences (because language is infinite), so that we could directly read off the information. Instead, we have to approximate the information by looking at spans of words rather than entire sentences. The spans usually consist of two words (bigrams) or three words (trigrams). For instance, the sentence "The

nice pig likes John" contains the trigrams "The nice pig", "nice pig likes", and "pig likes John".

A list of *n-grams* (for some natural number *n*) together with their probabilities derived from a corpus (by counting their occurrences) constitutes an instance of a Markov model and is called a *language model*. Using a bigram or trigram language model, a speech analysis system can now compute that it is much more probable that "pig" will occur than "big" in the context "Another nice _ likes Mary" (even if this particular sentence was not part of the training corpus).

The noisy-channel model and n-gram models can be applied in a large variety of tasks. One of the earliest considerations of automatic machine translation refers to the noisy-channel model and cryptography: "When I look at an article in Russian, I say 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.'" (Weaver 1949; also see Section 3.1 below). According to this view, an extreme form of "perturbation" has occurred when an English source text is affected in such a way that it comes out as a Russian text. The same statistical methods above can be used in reconstructing the source, i.e., in translating Russian "back" to English.

The model can also be used for linguistic analyses, such as part-of-speech tagging. Here, the "perturbed" signal is words in a text, and we use the model to reconstruct the corresponding parts of speech of the words as the assumed source signal. For this kind of task, *Hidden Markov Models* (HMM) are used, an extension of n-gram (Markov) models. In a HMM, multiple paths can lead to the same output, so that the functioning of the automaton can be observed only indirectly, whence the term "hidden". For example, the task of a PoS tagger is to deliver the correct PoS tags of some input words. The "visible" information is then the sequence of input words, and the "hidden" information is the corresponding sequence of PoS tags (which we are interested in). HMMs combine the probabilities of the visible information with those of the hidden information, so that we can reconstruct the PoS tag sequence from the word sequence.

The shared assumption of n-gram-based approaches is that linguistic phenomena of different kinds depend, to a large extent, on "some" local context information only. For instance, one might assume that for modeling word-related phenomena such as parts of speech, only the current word and its immediate neighbours, or only up to two words to the left, are "relevant"; for modeling constituent-related phenomena, only up to two constituents to the left are relevant, etc. (this is called a "Markov assumption", here of third order, because probabilities depend on three items: the current word/constituent and its two neighbours). These assumptions seem justified in the light of the cognitive, iconic

Proximity Principle: "Entities that are closer together functionally, conceptually, or cognitively will be placed closer together at the code level, i.e., temporally or spatially" (Givón 1991, 89). Certainly, it is true that there are many linguistic phenomena that involve "long-distance dependencies", such as circumfixes in morphology and agreement or wh-movement in syntax -- however, the idea is that a "sufficient" amount of data can be modeled successfully by looking at local context only. For instance, Marshall (1983) observes that in general a local context of two words provides enough information to yield "satisfactory results" in the automatic assignment of part-of-speech tags (see Section 3.2.4 below).

2.1.4. Supervised and unsupervised learning

(See articles 41 and 42.) We already mentioned that a probabilistic CFG can be trained on the basis of annotated or raw corpus data. If, due to annotations, the parameters of the probabilistic model are directly observable in the training corpus, we are dealing with "supervised" learning; if the parameters are not observable, it is called "unsupervised" learning.

In the scenarios described above, we measured "similarity" between the observed signal and source signal candidates. To do this, we need corpora that are annotated with the relevant information. A program that should learn to correct spelling mistakes needs a training corpus of real text whose spelling errors are annotated with the corrected version; a PoS tagger, which should learn to automatically assign parts of speech to words, needs a training corpus whose words are annotated with PoS tags; and so on. Training on annotated corpora is called "supervised learning", and the task that the program has to learn is called a "classification task": the system classifies each input according to a predefined set of classes (categories).

In unsupervised learning, the system is confronted with unannotated data only. It then tries to group (to cluster) the data in such a way that the clusters are distinguished from each other by characteristic combinations of certain features. What are these features? Suppose, for instance, we want to learn PoS tagging by unsupervised learning. That is, we feed the system with unannotated text and expect it to somehow come up with clusters that (more or less) correspond to linguistic PoS categories. The basic idea is that the system can learn these classes simply by comparing distribution in texts (thus implementing Firth's much-cited slogan "You shall know a word by the company it keeps!", Firth 1957), since we know that parts of speech reflect/encode distributional similarity. For instance, all words of category "noun" show similar distribution and occur in similar contexts: contexts like "the ... is" or "a ... has" are characteristic of nouns. In this scenario,

the features that the learning algorithm uses would be the word's neighboring words. The system then starts by assigning random probabilities (instead of probabilities derived from annotated corpora), then iteratively adjusts these values to maximize the overall probability of the entire text. It could thus learn, e.g., that "in" and "on" are members of the same cluster since they share many leftward and rightward neighbors. Of course, the system will not be able to guess the labels of the clusters (like "preposition" or "noun"), nor even the number of clusters that it is supposed to identify.

To sum up, supervised and unsupervised learning are two different approaches to machine learning: a system is fed (trained) with example data and derives generalizations from the data; after the training phase, the system can be applied to new, hitherto unseen data and classify (or cluster) this data according to the generalizations. To date unsupervised systems cannot in general learn highly fine-grained tagsets, and do not yet achieve the performance levels of systems that have been trained on annotated data.

2.1.5. Sparse data

The features that a system makes use of during training are obviously crucial to the success of the enterprise: there is probably no corpus so large that simple features like neighboring words would provide enough evidence for learning algorithms. In fact, there are no corpora so large that all possible kinds of phenomena would really occur in them, or that their frequencies would be high enough for learning algorithms. This is called the *sparse-data problem*. To overcome this problem, more abstract features have to be used, which generalize over relevant properties of the underlying words. For instance, in the clustering task above, the training features might consist of affixes of *n* characters length rather than full word forms. The learning algorithm would then cluster and generalize over strings like "-able" or "-ally". Another kind of generalization can be provided by annotations. No generalization, however, can compensate the sparse-data problem completely.

Automatic selection of suitable training features is an important topic: the system itself aims at finding the optimal subset of features from among a predefined set; the predefined set often consists of linguistically-motivated features but also includes superficial features like word length or position (in the text).

Another method to cope with the sparse data problem is a technique called *smoothing*: smoothing decreases the probabilities of all seen data, then

redistributes the saved amount (probability mass) among the unseen data, by assigning them very small probabilities of equal size.

2.1.6. Corpus annotation

As we have seen, annotated corpora are a vital prerequisite of supervised learning, and, in view of the sparse data problem, of unsupervised learning as well: corpora provide suitable abstraction layers over the training data. Annotated corpora are similarly important for theoretical linguists, and resources of this kind are being successfully exploited by both camps (see article 17). Annotations encode diverse kinds of information, such as part of speech (article 25), lemma (article 26), word sense (article 27), syntax trees (article 17), etc. Applied computational linguistics is also interested in broader regions of text, like paragraphs or entire texts. Annotations can thus also encode, e.g., the logical document structure, by marking regions as "header" or "footnote", or the content structure of texts, by labeling regions as "introduction" or "conclusion". A further example is alignment of comparable regions from different (possibly multilingual) sources.

Such corpora are usually first annotated manually, and subsequently exploited for training. Further annotation can then be performed (semi-)automatically, by applying the system that has been trained on the first data. An example bootstrapping approach of this kind is the PoS tagger CLAWS, which is presented in Section 3.

Due to the interest in annotated corpora, a lot of work in computational linguistics has been devoted to the development of corpus tools, such as tools to assist the annotator in the annotation, or search tools that support the use of linguistically-motivated search operators like "linear precedence" or "structural dominance", the basic relations of theoretical syntax. In this area, computational and corpus linguistics completely overlap in their interest in tools and methods.

2.1.7. Evaluation

As with linguistic theories, trained systems can be evaluated by testing their performance on new, unseen data, i.e., by evaluating whether the predictions of the theory or system fit the unseen data. Of course, computational linguistics aims at automatic evaluation, since systems nowadays deal with large amounts of data and are supposed to handle unrestricted texts. There are different methods of evaluation. Firstly, a supervised-learning system can be trained and evaluated on the same type of data. In this case, only part of the data (e.g., 90%) is used for training (and system development), and the remaining data is used as test (=

evaluation) data. It is crucial that neither the system nor the system's developer make any use of the test data other than for the very final evaluation, otherwise, the evaluation cannot be used as an indication of how well the system performs on genuinely new data. For testing, the trained system is run on the test data with the annotations stripped off. The output of the system is then compared with the original annotation, and the system's performance is computed in terms of measures such as *precision* and *recall*. Precision measures how accurate the system's predictions are; for instance, if a certain system assigns 10 NP chunks, and 8 of them are correct, then the system's precision equals 8/10 = 0.8. If the system should actually have marked 20 NP chunks but it found only 8 of them, then its recall equals 8/20 = 0.4. The measures thus encode both the fact that if this system marks an NP, it is usually correct, and the fact that it misses many of the NPs. A measure combining precision and recall is *F-score*, the harmonic means of both; in our example: $2*(0.8*0.4)/(0.8+0.4) = 5.33$.

If the performance of several systems is to be compared, a gold standard corpus is usually used as the test corpus (see Section 1.3), such as the PennTreebank (Marcus/Santorini/Marcinkiewicz 1993). A disadvantage of this type of evaluation is that the outputs of the systems are often not easily mapped to the gold standard, for example due to theory-dependent discrepancies, with the result that the performance of a system might actually be better than the evaluation outcome suggests. With unsupervised-learning systems especially, the discrepancies between a linguistically-motivated gold standard and the clusters of a system can be enormous.

For certain tasks like Information Retrieval, Machine Translation (see article 50) or Automatic Text Summarization (article 65), no gold standard is immediately available for evaluation, and with these tasks, it is hard to define "the best solution". Depending on the user and the situation, a range of different document selections might count as optimal for a specific Information Retrieval task. Similarly, there are always many alternative ways of translating or summarizing a text, and a gold standard would have to account for all these alternatives. In these cases then, manual inspection of the system output is often deemed necessary, and (subjective) measures like "quality" or "informativeness" are used.

The evaluation methods addressed so far are instances of *intrinsic* evaluation, because the performance of the system is measured by evaluating the system itself. Another method is *extrinsic* evaluation, in which the system is embedded in another application and the overall performance is measured. For instance, in order to evaluate a summarization system, one first asks people to assess the relevance of particular documents to a certain topic, or to answer document-

related questions. It can then be measured whether these people perform similarly when confronted with automatically-generated summaries of the documents instead of the full text; if so, the summarization system does a good job.

Evaluation nowadays plays an important role in applied computational linguistics, and researchers who develop new methods or tools are expected to provide the results of standardized evaluations alongside presentation of the system. A series of conferences focusing on evaluation issues has been initiated in the U.S., starting with MUC (Message Understanding Conference, since 1987; see also article 28) and TREC (Text REtrieval Conference, since 1992). These conferences in fact consist of competitions: each year, the conference organizers define a set of specific tasks, such as "for each open class word in the text, determine its sense according to the WordNet (Fellbaum 1998) synsets". They also provide researchers with relevant training data, well in advance of the conference, so that the system developers can tune and adapt their systems to the task and data. During the conference, the organizers present comparative evaluations of the systems that have been "submitted" to the conference.

To conclude this section, let us quickly review the deficiencies of handcrafted systems identified at the beginning of this section and compare them to the outcomes of statistical methods.

(i) Scalability: usually, statistical methods scale up well; if the training data is too small and does not include (enough) instances of certain phenomena, then the training corpus has to be enlarged, but the overall training method can be kept unchanged. (ii) Robustness: input data that does not meet the expectations of the system can be handled by smoothing, which assigns very low probabilities to unseen (and, hence, unexpected) data. However, scalability as well as robustness of a system are often sensitive to the types and domains of text that the system is confronted with: if the system has been trained on a certain text type and domain, its performance usually suffers if it is fed with texts of other types and domains. (iii) Ambiguity: ambiguities such as "The man saw the woman with the telescope" actually require some sort of semantic, contextual or even world knowledge to resolve them. Useful approximations of such knowledge are however provided by statistical methods that take lexical, collocational information into account: certainly the lemma "telescope" co-occurs with the lemma "see" more often than with "man"; similar preferences can be derived from (large) treebanks. (iv) Portability: usually the techniques applied in statistical approaches are language-independent, so portability is not an issue, in principle; the features that are used

in training must be carefully selected however; e.g., it makes no sense to use affix-like features in an isolating language like Chinese.

2.2 Statistical computational linguistics and corpora

The previous sections have shown clearly that texts and annotated corpora play a predominant role in application-oriented and statistical computational linguistics. They are a sine-qua-non condition both in training and in evaluating statistical systems. Linguistic information in the form of annotation is usually part of the training data (other kinds of resources, such as WordNet (Fellbaum 1998), often provide additional information). Corpus annotation and corpus tools are thus a concern of both corpus and computational linguistics and annotation-related research and methods can often be attributed to both disciplines. Similarly, just as in corpus-based linguistic research, techniques in computational linguistics that make use of corpus frequencies are faced with the fact that corpora are finite samples and, to generalize from such samples, statistical inference is needed (see article 38), and methods like n-gram approximations and smoothing have to be applied.

As already mentioned, at opposite extremes of computational and corpus linguistics, the ultimate goals of corpus exploitation do however diverge in that the features that turn out to be useful for language models are not seen as interesting *per se* by certain domains of corpus-driven computational linguistics. Indeed, many researchers think it would be most desirable to let the algorithms specify (define) the features fully automatically -- and some of these researchers only care about the performance of the system rather than the features used by the system. Unfortunately for them, corpus data is too restricted to provide enough evidence for all sorts of conceivable features that an algorithm might come up with (and, probably, computer capacities also set limits to such an enterprise). Therefore, a set of features has to be predefined that the algorithms can choose from. These sets often contain both linguistic and non-linguistic features and, very often, simple non-linguistic features like the average word or sentence length, the standard deviation of word/sentence length, the number of periods, question marks, commas, parentheses, etc., are successfully exploited by learning algorithms. These features are certainly reflections of interesting linguistic phenomena, e.g., the number of commas can give hints about the syntactic complexity of the sentences. However, the connection between the features and the linguistic properties is rather loose, so that corpus linguists would be not interested so much in such features. They usually very carefully select the features that they are interested in.

In contrast to theory-driven computational linguistics, corpora are mainly used quantitatively. The knowledge encoded in annotations thus becomes part of any language model derived on the basis of the data. However, the development of statistical models can also involve qualitative analysis: for example, during the development phase, researchers will quite carefully inspect the data (and its annotation) in order to identify an optimal set of discriminative features for use in training. Similarly, evaluations are usually accompanied by more or less detailed *error analyses*, to facilitate better understanding of the weaknesses and shortcomings of the system; for this, researchers will scrutinize that part of the test data that most often causes the system to fail.

We now proceed to Section 3, which presents a short historical overview of the origins and early development of computational linguistics, focussing on early machine translation and the evolution of part-of-speech tagging. This area neatly illustrates the application of hand-crafted rules in the first generations of PoS taggers, which were later supplemented and finally replaced by corpus-driven techniques.

3. Computational linguistics: a brief history

3.1. The emergence of computational linguistics: first steps in machine translation

The origins of computational linguistics can be traced back (cf., e.g., Hutchins 1997, 1999) to the American mathematician Warren Weaver, director of the Natural Sciences Division of the Rockefeller Foundation, who during World War II, became acquainted with the development of electronic calculating machines and the application of statistical techniques in cryptography. In July 1949, he wrote his famous memorandum, "Translation", suggesting that automatic translation might be feasible (Weaver 1949, see Section 2.1.3). At that time, early experiments in machine translation had already been conducted on the basis of word-by-word translation of scientific abstracts. The results of these translations were of course far from satisfactory -- as will be obvious to anyone with basic linguistic knowledge.

In his memorandum, Weaver made four proposals as to how to overcome the problems of word-by-word translation, two of which we address here because they refer to information that can be extracted from raw texts. (The other proposals relate to the "logical character" of language and the existence of language universals.) Weaver's first proposal concerned the disambiguation of word meaning:

> "If one examines the words in a book, one at a time through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of words. "Fast" may mean "rapid"; or it may mean "motionless"; and there is no way of telling which. But, if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then, if N is large enough one can unambiguously decide the meaning."

The second proposal concerned the application of cryptographic methods (Shannon 1948), based on "frequencies of letters, letter combinations, intervals between letters and letter combinations, letter patterns, etc. ...".

In both proposals, information is gained from texts and contexts: in the first proposal, a word can be disambiguated by examining its *current* context. The second proposal suggests that such tasks might also be achieved by examining the *usual* context of a word, i.e., the contexts in which the word commonly (most frequently) occurs. The n-gram example in Section 2.1.3, disambiguating "big/pig", implements these proposals.

3.2. The emergence of symbolic methods in computational linguistics

Returning to the evolution of machine translation, the next major event after the Weaver memorandum was the publiction of the ALPAC report in 1966. At this time, most US funding for computational linguistics had gone into projects on machine translation. In 1964, a committee of experts were asked to judge whether this funding was justified, and in their final report (ALPAC 1966) they came to the devastating conclusion that none of the applications developed so far were satisfactory, and that employing human translators would not only yield better results but would also be cheaper. The report suggested abandoning the funding of scientific research on machine translation, but, instead, encouraged the support of fundamental research in computational linguistics in general. In particular, the ALPAC report recommended:

> "(1) basic developmental research in computer methods for handling languages, as tools to help the linguistic scientist discover and state his generalizations, and as tools to help check proposed generalizations against data; and (2) developmental research in methods to allow linguistic scientists to use computers to state in detail the complex kinds of theories (for example, grammars and theories of meaning) they produce; so that the theories can be checked in detail. " (ALPAC 1966, vi)

Clearly, the main role of computational linguistics was seen as assisting linguists in the formalization and assessment of linguistic theories, very much as described in Section 1. The ALPAC report had a tremendous impact on the course of computational linguistics research in the following years, causing a major change in the focus of research: a shift from mainly statistical methods to rationalist approaches, using symbolic, linguistic features, rather than numerical values, such as probabilities. More emphasis was put on the analysis of the nature of the underlying categories that constitute natural language, and on their interaction, introducing different levels of categories and structures, such as simple part-of-speech (PoS) tags or complex syntactic structures.

In our presentation here, we focus on PoS tagging. We will see that the very first automatic PoS taggers implemented versions of Weaver's first proposal, in the form of context rules. These rules were created manually, but later taggers derived them from annotated corpora, thus implementing Weaver's second proposal (applying his proposal to annotations rather than words). We will further see that people then started to use annotated corpora to evaluate the performance of their system.

3.2.1. The TDAP parser

Probably the first automatic PoS tagger was implemented by Zellig S. Harris in 1958/59 (cf. Jurafsky/Martin 2000, 317). The tagger was designed as a preprocessing component of the TDAP parser ("Transformation and Discourse Analysis Project", Harris 1962; reimplemented as "Uniparse" by Joshi/Hopely 1998). In this architecture, the tagger first assigns to each word a set of tag candidates, which are looked up in a lexicon. Next, a series of rules are run, eliminating tags that are incompatible with the local context, thus implementing Weaver's first proposal. For instance, the tag "V" (verb) is deleted from the set of candicates if the word is followed by the preposition "of", although a set of designated verbs, such as "think, consist", are exempted from this rule. All in all, the system comprised 14 hand-written rules for PoS disambiguation, which were run in cycles until no further disambiguation was possible.

On top of these PoS-disambiguation rules, another series of cascaded rules were applied to the previously-identified PoS tags, to parse the sentence. Thus, the tagger functioned as a preprocessor for the parser, by introducing a first layer of abstract linguistic categories (parts of speech) to encode syntax-relevant information in the text. The parser could recognize basic, non-recursive syntactic constituents, such as base NPs and PPs. The parser rule for NPs, for instance, which was applied from right to left, recognized tag sequences of the form "N A*

(T)", using a longest-match strategy. That is, it first allocated the head noun (N), next, arbitrarily many adjectives (A*) preceding that head noun, and, finally, an optional determiner (T). An example TDAP parse is shown in (1) (taken from Bangalore 1997, 21); [ ] indicate NPs, { } verb sequences, and ( ) adjuncts.

(1) [Those papers] {may have been published} (in [a hurry]).

The TDAP parser could not handle unknown words. One of the reasons was certainly the limited computational capacities available at that time, both in terms of storage size and processing speed. Implementations could at that time be fed by small amounts of data only, so there was no need to process large amounts of free text, featuring unknown words.

3.2.2. The CGC tagger

The next step in the evolution of PoS tagging was made by Klein/Simmons (1963), who developed a tagger in the context of a natural language question-answering system. The tagger, called "CGC" (computational grammar coder), assigned PoS tags based on suffixes rather than words. The algorithm first looked up each word in dictionaries of approximately 400 function words (articles, prepositions, pronouns, ...) and 1,500 irregular forms (nouns, verbs, adjectives). For the remaining words, tag candidates were assigned according to the suffix of each word. For instance, the test "Suffix Test 1" handled English inflection: it marked, e.g., words ending in "-ies", such as "nationalities", as NOUN/VERB (plural noun or 3rd person singular verb). The algorithm would then replace the presumed suffix "-ies" by "y" ("nationality") and perform another test, "Suffix Test 2", on the new form. "Suffix Test 2" included information about derivational suffixes, so that the newly-created word ending in "-ity" would be recognized as NOUN. After running a series of such tests, the individual results were intersected; in our example, "Suffix Test 1" (NOUN/VERB) and "Suffix Test 2" (NOUN) would be resolved to NOUN as the final tag. For remaining ambiguities, or words that had not yet been assigned some tag, the "Context Frame Test" could delete, or add, tag candidates, based on a hand-crafted list of about 500 permissible tag sequences. As with the above disambiguation rules, the Context Frame Test implements Weaver's first proposal.

Besides the fact that the CGC tagger needed small lexica only, it had the enormous advantage of being robust: regardless of the input, the tagger always came up with some analysis. Since it was the first system that was actually able to deal with unrestricted text, it now made sense to evaluate the CGC tagger, by tagging previously-unknown text. Klein/Simmons (1963, 344) report that they

tagged "several pages" of a children's encyclopedia, and the tagger "correctly and unambiguously tagged slightly over 90 per cent of the words". This is a surprisingly good result; however, it is not fully clear from their paper whether they or not had used the evaluation data in the development of the system. If so, this would mean that their system was optimized for this data and could not be expected to yield similarly good results for unseen texts. Moreover, since all of the knowledge resources of the system (dictionaries, suffix tests, context frame test) were hand-crafted, it is not obvious whether the system would scale up, i.e., be extensible to large-scale texts.

3.2.3. TAGGIT

Continuing the work of Klein/Simmons, TAGGIT (Greene/Rubin 1971) was the first tagger that was actually applied to large-scale texts, namely the Brown Corpus (Kucera/Francis 1967; see article 22). Like the CGC tagger, TAGGIT used a dictionary, with about 3,000 entries, then applied a suffix list of about 450 strings, followed by a filtering through 3,300 context frame rules (which play the role of the Context Frame Test of the CCG tagger). In comparison to the CGC tagger, TAGGIT used a more fine-grained tagset (82 tags, as opposed to 30 CGC tags), and the suffix list was derived from lexico-statistics of the Brown Corpus. Crucially, and in contrast to the previous approaches, the TAGGIT context rules were acquired semi-automatically: the tagger (without context frame rules) was run on a subset of 900 sentences from the Brown Corpus, and for all ambiguous cases the correct tag was determined manually; a program was then run to produce and order all of the context frame rules that would have been needed for the disambiguation. According to Kucera/Francis (1982), TAGGIT correctly tagged approximately 77% of the Brown Corpus. In the 1970s, TAGGIT was used to annotate the entire Brown Corpus; any words that did not receive a unique tag from TAGGIT were manually disambiguated. The Brown Corpus is therefore not only the first large-scale *electronic* corpus, and the first *annotated* corpus, but also the first corpus which was ever annotated by an *automatic* tagger.

3.2.4. … and back to statistics: CLAWS

The Brown Corpus, which consists of texts of American English, was soon complemented by a corresponding corpus of British English, the LOB corpus (Leech/Garside/Atwell 1983). Similarly to the Brown Corpus, the LOB corpus was intended to be enriched by PoS tags. For this task, Leech and colleagues benefitted from the annotated Brown Corpus and the TAGGIT program itself. TAGGIT had an accuracy of 77%; this means that, assuming an average sentence

length of 19 words, every sentence would contain, on average, 4 ambiguous tags, which had to be manually post-edited. This heavy load of human intervention was obviously a serious problem for further large-scale annotation. To improve the performance of TAGGIT, the LOB group developed a program that applied statistical methods, known as the "tagging suite", later called "CLAWS" ("Constituent-Likelihood Automatic Word-Tagging System", Marshall 1983, Leech/Garside/Atwell 1983). CLAWS inherited much from TAGGIT: it comprised a lexicon of 7,000 entries, derived from the tagged Brown Corpus, plus a list of 700 suffixes, and rules for certain words. The important innovative aspect of CLAWS was the implementation of the context frame rules, by means of a statistical program called the Tag Selection Program (Marshall 1983). The program used PoS-bigram frequencies computed from the Brown Corpus. Given a sequence of ambiguous tags, which had been assigned in previous steps, the program first enumerated all possible disambiguated tag sequences. Next, it computed the probability of the entire sequence as the normalized product of all of the individual bigram frequencies occurring in the sequence. Finally, the tag sequence that maximized the probability was chosen.

To give an example: suppose lexicon lookup and suffix rules resulted in the following ambiguous tag sequence (example from Marshall 1983):

(2)

Henry	NP
likes	NNS VBZ
stews	NNS VBZ
.	.

This sequence compiles into four disambiguated tag sequences:

(3)
a. NP NNS NNS .
b. NP NNS VBZ .
c. NP VBZ NNS .
d. NP VBZ VBZ .

The frequency for each tag bigram ("NP NNS", "NNS NNS", "NNS VBZ", etc.) is computed from the Brown Corpus; e.g., freq(NP NNS) = 17, freq(NNS NNS) = 5, freq(NNS .) = 135, etc. The probability of the complete sequence "NP NNS NNS ." (3a) is then computed as the product of the individual bigrams, i.e., 17∗5∗135 = 11,475, divided by the sum of the values for all possible sequences (38,564), resulting in the probability P("NP NNS NNS .") = 0.3.

Marshall observed that, in general, bigram frequencies yielded satisfactory results. In certain cases, however, bigrams could not encode enough information:

for instance, in sequences of the form "verb adverb verb" (as in "has recently visited"), the second verb was often incorrectly analyzed as past tense ("VBD") instead of the correct past participle ("VBN"). To extend the context window, selected tag triples (or trigrams) were added to the program to avoid such errors. For certain cases, it proved useful to exploit the probability of a word-tag assignment, derived from the Brown corpus, as an alternative method of calculating the most probable tag. CLAWS has been used to tag the entire LOB Corpus and achieved an overall success rate of between 96% and 97% (which is very close to the performance of today's taggers).

The development we have traced, by looking at part-of-speech tagging algorithms, clearly shows how theory-driven methods were applied to make explicit the information contained in a corpus, through analysis of linguistic expressions in terms of more abstract linguistic categories and structures. This information was exploited again, to train statistical methods, but on the basis of yet more abstract linguistic structures than the original surface-oriented methods. At the same time, the annotated corpus was also used to evaluate the performance of the automatic systems.

4. Conclusion

In summary then, the history of computational linguistics started out with a strong focus on statistics, which were computed on raw, unanalyzed texts, then moved on to research on theoretical frameworks and theory-driven, more linguistically-informed methods (by introducing linguistic categories like PoS tags), and, finally, came back again to use of statistics on annotated texts. Needless to say not all research in computational linguistics fits exactly into this picture.

As we have seen, computational linguistics started out as an application-focused task, namely automatic translation, and from there evolved into a broad research area, comprising different research methodologies (corpus- and theory-driven), fundamental and application-oriented research, and involving different linguistic disciplines. Interestingly, the methods that were judged to be useless in the beginning were later successfully revived, but combined with more substantial, linguistically-informed, formal models of language. Today, statistical approaches are prevalent in many areas of computational linguistics; e.g., the majority of approaches presented at the most important conferences of computational linguistics -- most prominently, the conferences of the Association

for Computational Linguistics (ACL, EACL, NAACL), or COLING, the International Conference on Computational Linguistics -- are oriented towards statistical language modeling. Why are statistical methods nowadays so successful, in contrast to earlier attempts in machine translation, and in contrast to purely-symbolic approaches? Here are some possible reasons: firstly statistics can these days draw information from annotated texts, which means that the input provides relevant information in a focused, condensed way. Secondly, the amount of data available for training has increased immensely. Thirdly, and in contrast to symbolic approaches, statistical methods are robust and can easily deal with defective input. Martin Kay, one of the pioneers of computational linguistics, adds another reason (e.g., Kay 2004): what was missing from the early approaches, and is indeed still missing today, is world knowledge. For successful language understanding, we need knowledge about objects in the world, about their relations to other objects, about the knowledge of the speaker and hearer about these objects, about their mutual beliefs, etc. Such knowledge is in part encoded in resources like WordNet, however, these resources are rather small and confine themselves to relations that are linguistic in nature, such as lexical relations of synonymy or hypernymy. They do not encode world knowledge, such as "in restaurants, meals are served by waiters". The representation of world knowledge is the domain of Artificial Intelligence. The modeling of world knowledge and commonsense reasoning is however a difficult and complex task, and as of today there are no large-scale implementations that would allow for practical applications in computational linguistics. Statistical methods can derive world knowledge from corpora to a certain extent. For instance, words like "restaurants, meals, waiters" will often co-occur in narrow contexts, if enough data is available that contain these words, and thus statistics can be used as a "poor man's" substitute for underlying world knowledge. Similarly, ambiguities such as "The man saw the woman with the telescope" can be resolved by collocational information derived from corpora: certainly the lemma "telescope" co-occurs with the lemma "see" more often than with "man".

These days then, we see an increasing tendency to rely on a mixture of both linguistic knowledge -- whether explicitly encoded in the form of rules (e.g., tagger or grammar rules) or implicitly encoded in the form of annotated corpora -- and statistical methods.

5. References

Abney, Steven (1996), Statistical Methods and Linguistics. In: Klavans, Judith

/Resnik, Philip (1996).

Ades, Anthony/Steedman, Mark (1982), On the Order of Words. In: *Linguistics and Philosophy* 4, 517-558.

Ajdukiewicz, Kazimierz (1935), Die syntaktische Konnexität. In: *Studia Philosophica* 1, 1-27.

ALPAC (1966), *Language and Machines: Computers in Translation and Linguistics*. A Report of the Automatic Language Processing Advisory Committee (ALPAC), Division of Behavioral Sciences, National Research Council. Washington, DC: Publication 1416, National Academy of Sciences.

Antworth, Evan L. (1990), PC-KIMMO: a two-level processor for morphological analysis. In: *Occasional Publications in Academic Computing* No. 16. Dallas, TX: Summer Institute of Linguistics.

Baker, James K. (1979), Trainable grammars for speech recognition. In: Wolf, Jared J./Klatt, Dennis H. (eds.), *Speech communication papers presented at the 97th meeting of the Acoustical Society of America*. MIT, Cambridge, Mass, 547-550.

Bangalore, Srinivas (1997), *Complexity of Lexical Descriptions and Its Relevance to Partial Parsing*. PhD thesis, University of Pennsylvania, IRCS Report 97-10.

Bar-Hillel, Yehoshua (1953), A quasi-arithmetical notation for syntactic description. In: *Language* 29, 47-58.

Baum, L.E. (1972), An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov Processes. In: *Inequalities* 3, 1-8.

Beesley, Kenneth R./Karttunen, Lauri (2003), *Finite State Morphology*. CA: CSLI Publications.

Black, Ezra/Lafferty, John/Roukos, Salim (1992), Development and evaluation of a broadcoverage probabilistic grammar of English-language computer manuals. In: *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 185-192.

Bledsoe, Woodrow W./Browning, Iben (1959), Pattern Recognition and Reading by Machine. In: *Proceedings of the Eastern Joint Computer Conference*, 225-232.

Bobrow, Daniel G./Fraser, J. Bruce (1968), A phonological rule tester. In: *Communications of the Association for Computing Machinery (ACM)* 11, 766-772.

Bod, Rens (1998), *Beyond grammar: an experience-based theory of language*. Stanford, CA: CSLI Publications.

Bod, Rens/Hay, Jennifer/Jannedy, Stefanie (eds.) (2003), *Probabilistic Linguistics*. Cambridge, MA: MIT Press.

Bresnan, Joan, (ed.) (1982), *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press.

Carpenter, Bob (1992), *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press.

Charniak, Eugene (1993), *Statistical Language Learning*. Cambridge, MA: MIT Press.

Charniak, Eugene (1996), *Tree-bank Grammars*. Technical Report CS-96-02, Department of Computer Science, Brown University.

Chomsky, Noam (1956), Three models for the description of language. In: *IRE Transactions on Information Theory* 2, 113-124.

Chomsky, Noam (1957), *Syntactic structures*. The Hague: Mouton.

Chomsky, Noam (1959), On certain formal properties of grammars. In: *Information and Control* 2(2), 137-167.

Chomsky, Noam (1993), A minimalist program for linguistic theory. In: Hale, Kenneth/Keyser, Samuel Jay (eds.), *The view from building 20*. Cambridge, MA: MIT Press, 1-52.

Chomsky, Noam/Halle, Morris (1968) "The Sound Pattern of English". New York: Harper and Row.

Chomsky, Noam/Miller, George A. (1958), Finite state languages. In: *Information and Control* 1(2), 91-112.

Copestake, Ann (2002), *Implementing Typed Feature Structure Grammars*. CA: CSLI Publications.

Copestake, Ann/Flickinger, Dan/Sag, Ivan/Pollard, Carl (2005), Minimal Recursion Semantics: An introduction. In: *Journal of Research on Language and Computation*, 3(2-3), 281-332.

Dalrymple, Mary (ed.) (1999), *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. Cambridge, MA: MIT Press.

Davis, H./Biddulph R./Balashek, S. (1952), Automatic recognition of spoken digits. In: *Journal of the Acoustical Society of America* 24(6), 637-642.

Dempster, Arthur/Laird, Nan/ Rubin, Donald (1977), Maximum likelihood from incomplete data via the EM algorithm. In: *Journal of the Royal Statistical Society*, Series B, 39(1), 1-38.

Earley, Jay (1970), An efficient context-free parsing algorithm. In: *Communications of the Association for Computing Machinery* 13(2), 94-102.

Fellbaum, Christiane (ed.) (1998), *WordNet An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Firth, John Rupert (1957), A Synopsis of Linguistic Theory, 1930-1955. In: *Studies in Linguistic Analysis*. Special volume of the Philological Society. Oxford: Basil

Blackwell, 1-32.

Givón, Talmy (1989), *Mind, Code and Context: Essays in Pragmatics*. Hillsdale, NJ: Erlbaum.

Greene, Barbara B./Rubin, Gerald M. (1971), *Automatic grammatical tagging of English*. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island.

Harris, Zelig S. (1962), *String Analysis of Sentence Structure*. The Hague: Moulton.

Hutchins, John (1997), First steps in mechanical translation. In: Teller, Virginia/Sundheim, Beth (eds.), *Proceedings of MT Summit VI: past, present, future*. San Diego, California, 14- 23.

Hutchins, John (1999), Warren Weaver memorandum: 50th anniversary of machine translation. *MT News International* 8(1) (= issue 22), 5-6.

Johnson, C. Douglas (1972), *Formal Aspects of Phonological Description*. The Hague: Mouton.

Johnson, Mark (1988), *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes No. 16. Stanford: CSLI.

Joshi, Aravind K. (1985), Tree adjoining grammars: How much contextsensitivity is required to provide reasonable structural descriptions?. In: Dowty, David R./Karttunen, Lauri/Zwicky, Arnold (eds.), *Natural Language Parsing. Cambridge University Press*, 206-250.

Joshi, Aravind K./Hopely, Philip D. (1998), A Parser from Antiquity. In: Kornai, András (ed.), *Extended Finite State Models of Language*. Cambridge University Press.

Joshi, Aravind K./Levy, Leon S./Takahashi, Masako (1975), Tree adjunct grammars. In: *Journal of the Computer and System Sciences* 10(1), 136-163.

Jurafsky, Daniel S./Martin, James H. (2000), *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, NJ: Prentice Hall.

Kaplan, Abraham (1950), *An experimental study of ambiguity and context*. Santa Monica: The RAND Corporation. Reprinted in: *Mechanical Translation* 2, 39-46 (1955).

Kaplan, Ron M./Bresnan, Joan (1982), Lexical-Functional Grammar: A Formal System for Grammatical Representation. In: Bresnan, Joan (1982).

Kaplan, Ronald M./Kay, Martin (1981), *Phonological rules and finite-state transducers*. Paper presented to the Winter Meeting of the Linguistic Society of America, New York.

Kaplan, Ronald M./Kay, Martin (1994), Regular Models of Phonological Rule

Systems. In: *Computational Linguistics* 20(3), 331-378.

Karttunen, Lauri (1983), KIMMO: a general morphological processor. In: *Texas Linguistic Forum* 22, 163-186.

Kasami, Tadao (1965), *An efficient recognition and syntax-analysis algorithm for contextfree languages*. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

Kasper, Robert T./Rounds, William C. (1986), A logical semantics for feature structures. In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*. Morristown: ACL.

Kay, Martin (2004), Introduction. In: Mitkov, Ruslan (ed.), *The Oxford Handbook of Computational Linguistics*, xvii-xx.

King, Paul (1989), *A logical formalism for Head-Driven Phrase Structure Grammar*. Ph.D. thesis, University of Manchester.

King, Paul John (1994), *An Expanded Logical Formalism for Head-Driven Phrase Structure Grammar*. Arbeitspapiere des SFB 340, University of Tübingen.

Klavans, Judith L./Resnik, Philip (eds.) (1996), *The Balancing Act: Combining Symbolic and Statistical Aprroaches to Language*. Cambridge, MA: MIT Press.

Kleene, Stephen Cole (1956), Representations of Events in Nerve Sets and Finite Automata. In: Shannon, Claude E./McCarthy, John (eds.), *Automata Studies*. Princeton, NJ: Princeton University Press, 3-41.

Klein, Sheldon/Simmons, Robert F. (1963), A computational approach to grammatical coding of English words. In: *Journal of the ACM* 10(3), 334-347.

Koskenniemi, Kimmo (1983), *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. University of Helsinki, Department of General Linguistics, Publication No. 11.

Kroch, Anthony S./Joshi, Aravind K. (1985), *The Linguistic Relevance of Tree Adjoining Grammar*. Technical Report, MS-CIS-85-16, University of Pennsylvania.

Kucera, Henry/Francis, Nelson W. (1967), *Computational Analysis of Present-Day American English*. Providence, R.I: Brown University Press.

Leech, Geoffrey/Garside, Roger/Atwell, Eric (1983), *The Automatic Grammatical Tagging of the LOB Corpus*. Newsletter of the International Computer Archive of Modern English (ICAME) 7, 13-33.

Manning, Chris/Schütze, Hinrich (1999), *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Marcus, Mitchell P./Santorini, Beatrice/Marcinkiewicz, Mary Ann (1993), Building a large annotated corpus of English: The Penn Treebank. In: *Computational Linguistics* 19, 313-330.

Marshall, Ian (1983), Choice of grammatical word-class without global syntactic analysis: tagging words in the LOB corpus. In: *Computers and the Humanities* 17, 139-150.

Mosteller, Frederick/Wallace, David L. (1964), *Inference and Disputed Authorship: The Federalist*. New York: Springer.

Oswald, Victor A. (1952), *Microsemantics*. Paper presented at the MIT Conference on Mechanical Translation.

Page, Ellis B. (1967), Statistical and Linguistic Strategies in the Computer Grading of Essays. In: *Proceedings of COLING (Conference Internationale Sur Le Traitement Automatique Des Langues)*.

Paroubek, Patrick/Schabes, Yves/Joshi, Aravind K. (1992), XTAG -- A Graphical Workbench for Developing Tree-Adjoining Grammars. In: *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP)*, 223-230.

Pollard, Carl/Sag, Ivan A. (1994), *Head-Driven Phrase Structure Grammar*. Stanford: CSLI, and Chicago: University of Chicago Press.

Rabin, Michael O./Scott Dana (1959), Finite automata and their decision problems. In: *IBM Journal of Research and Development* 3(2), 114-125.

Savitch, Walter J./Bach, Emmon/Marsh, William/Safran-Naveh, Gila (eds.) (1987), *The Formal Complexity of Natural Language*. Studies in Linguistics and Philosophy No. 33. Dordrecht: Reidel.

Shannon, Claude E. (1948), A mathematical theory of communication. In: *Bell System Technical Journal* 27, 379-423 and 623-656.

Stabler, Edward (1997), Derivational minimalism. In: Retore, Christian (ed.), L*ogical Aspects of Computational Linguistics, LACL'96*. Berlin: Springer, 68–95.

Stede, Manfred (1992), The search for robustness in natural language understanding. In: *Artificial Intelligence Review* 6, 383-414.

Steedman, Mark (1985), Dependency and coordination in the grammar of Dutch and English. In: *Language* 61, 523-568.

Steedman, Mark (1996), *Surface Structure and Interpretation*. Linguistic Inquiry Monograph No. 30. Cambridge, MA: MIT Press.

Steedman, Mark (2000), *The Syntactic Process*. Cambridge, MA: MIT Press.

Tesnière, Lucien (1959), *Eléments de syntaxe structurale*. Editions Klincksieck.

Viterbi, Andrew J. (1967), Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Transactions on Information Theory* 13(2), 260-269.

Weaver, Warren (1949), *Translation*. Reprinted in: Locke, William Nash/Booth, Andrew Donald (eds.) (1955), *Machine translation of languages: fourteen essays*. New York: Wiley, 15-23. 15-23.

Younger, Daniel H. (1967), Recognition and parsing of context-free languages in time n^3. In: *Information and Control* 10(2), 189-208.

van Zaanen, Menno (2000), ABL: Alignment-Based Learning. In: *Proceedings of the 18th Conference on Computational Linguistics*, Volume 2.