From algorithms to generative grammar and back again

John Goldsmith
The University of Chicago

1. Introduction

A few words of warning. I was asked to offer a contribution to this meeting on the application of linguistic theory. I'm not sure I know enough about this question to provide any new insights into applications, but I have been working for several years on a project which involves language, linguistics, and computation, and which has some very applied sides to it: a system for automatically learning the morphological structure of a language, given a large enough sample of data from the language. Working on this project has changed in varying degrees my view of what linguistic theory is, could be, and should be, and so since this is my own personal experience that I will recount, I might have something to contribute to a more general discussion.

I have been interested for quite some time in the possibility and the implementation of computer systems that deal with natural language. Now, natural language is (in the opinion of linguists) the natural domain of linguists and of linguistics, and I have observed with great interest the successes that have occurred as linguistic ideas have been reincarnated in computational form, and have worried when linguistic notions have failed to make the transition to computational form. The remarks that follow are a series of reflections on why some of those transitions – from theory to computational application – have happened and why some have not. In the second half especially, these remarks are unquestionably (and unapologetically!) personal and idiosyncratic.

In the first part of the talk, I will review and explore some of the natural connections between generative grammar and computational implementation, connections that relate to the fundamental notion of *the algorithm*. Generative grammar is, more than it is anything else, a plea for the case that an insightful theory of language can be based on algorithmic explanation.

But there are a lot of challenges faced by anyone wishing to implement generative models computationally.

2. The algorithm as *explicans*

The first idea I would like to explore is the notion that the greatest changes in linguistics have occurred not through the discovery of new facts or theories, but through the rise and dissemination of new visions of what *counts* as an explanation in linguistics. In the final analysis, this is what keeps apart linguists

who come from separate camps. That is, it's not what one believes to be true or false about language that separates linguists so much as it is beliefs about what counts as an explanation that separates linguists. I will try to make that statement plausible by sketching the rise of a few such schools of thought in linguistics.

No one is more aware than we linguists are at the variety there have been of major approaches to the systematic study of language. The 19th century was dominated by the study of language as a system in time – in human history, in fact. What it meant to *explain* something (such as a word, a sound, or a language), in the context of 19th century linguistics, was to give a clear and detailed account of how it came to be, given an earlier account of the language and a pattern of how things changed over time. We can explain *why* French has the word *oui* for 'yes' while Occitan has *oc* for 'yes' if we understand that both developed out of Latin *hoc*, and in the North of France the final *k* was dropped, and the remaining *o-* had a encliticized *il* attached to it, *oïl* eventually evolving to modern-day *oui*. Linguistics came to understand the relationship of the European languages, and later the languages of other parts of the world, as part of a small number of genealogical trees.

The 20th century saw not just new ways of analyzing language, but new ways of offering explanations for things that were observed. Synchronic, structural linguistics came into being, often associated with de Saussure's teaching, and it offered a new kind of explanation. Take phonemics, the flagship theory within structural linguistics: it could offer an account of why a flap appears in American English in the word *Ítaly* but not the word *Itálian*, and this despite the fact that the words are closely related. The explanation lies in the organization of the environments appropriate for the two allophones, an explanation that is in no way dependent on a historical account of the language or the words. It is perhaps unnecessary to remind a gathering of linguists that this was linguistics' greatest contribution to the outside world: the discovery of a method of analysis of a cultural data (here, language), which linguists called phonemic analysis, which was both rigorous and sensitive to details at the human level, and which analyzed language in an ahistorical fashion. One of the by-products of the creation of the synchronic phonemic method was modern linguistics as we know it today.

Other kinds of explanation have been offered for linguistic structures during this period as well, notably *psychological* explanation and *sociological* explanation. Over the last several decades, linguists have tended to call themselves *functionalists* who believe that an explanation of a linguistic generalization must flow from a generalization about the psychological character and mechanisms of human beings; and while many linguists who are not functionalists would say that they are *describing* or *analyzing* the human mind, it is only the functionalists who see the flow of explanation – from explainer to explained, from *explicans* to *explicandum* – as coming *from* principles that are established using the methods

of the psychology profession. (That predilection has a long history, and was once known as *psychologism*.) Many prominent linguists subscribe to this point of view to one degree or another, I would say, including Talmy Givón, Scott DeLancey,[1] perhaps Susumu Kuno, on some days George Lakoff, Joan Bybee, Jack Hawkins, and many others.

Another form of explanation is offered by linguists who see language as a social phenomenon, and hence strongly influenced by the principles that govern the interaction of human individuals operating within social organizations. Such linguists have been driven to understand the differences between the way men and women speak, those between the ways adolescents and adults speak, and the relationship between the class structure in society and the engines of linguistic change, but more important (for my purposes) than the nature of the questions they pursue is the kind of explanation that they offer, explanation that is based on what we know about the way humans behave in the context of social norms and expectations.

But perhaps the most momentous shift in what counted as an explanation in linguistics arose at mid-century, and is generally associated with Noam Chomsky's revolutionary work. Chomsky championed the notion that what the linguist needed to offer as an explanation was a formal description of the linguistic facts, a description that was formal and detailed to a degree that no human intervention is needed to apply the description appropriate to the facts (which is to say, nothing more than mechanical means is necessary). What counted as an explanation, according to this new perspective, was an *algorithm* (though, oddly enough, I don't recall the term ever being used by Chomsky). This is an important notion, one whose history will matter to us, so we will take a look at it in a moment. But it was Chomsky's central idea, one that he put at the definitional core of what he called *generative grammar*: as Chomsky wrote in the introduction to *The Logical Structure of Linguistic Theory* (1975),

> conventional structuralist grammars or traditional grammars do not attempt to determine explicitly the sentences of a language or the structural descriptions of these sentences. Rather, such grammars describe elements and categories of various types, and provide examples and hints to enable the intelligent reader to determine the form and structure of sentences not actually presented in the grammar. Such grammars are written for the intelligent reader. To determine what they say about sentences one must have an intuitive grasp of certain

[1] Two linguists associated with the University of Oregon: in fact, their website notes, "Our department as a whole firmly believes that the patterns of language can ultimately be explained with reference to either cognitive functions of communication or to universals in the evolution of grammar, with the patterns of evolution themselves driven mostly by the cognitive functions of communication." See http://logos.uoregon.edu/

> principles of linguistic structure. These principles, which remain implicit and unexpressed, are presupposed in the construction and interpretation of such grammars. While perhaps perfectly adequate for their particular purposes, such grammars do not attempt to account for the ability of the intelligent reader to understand the grammar. The theory of generative grammar, in contrast, is concerned precisely to make explicit the "contribution of the intelligent reader," though the problem is not posed in just these terms. (8-9)

But *why? Why* is such a goal the best goal for an explanatory theory of language, or even just a reasonable goal? Coming up with an answer to that question involves going back in time to the development of the notion of the algorithm, and seeing what questions it was that its development was intended to solve, and what problems it did in fact solve.

3. Generative model as algorithm

The term *algorithm* in modern English derives from the Middle English *algorism*, itself deriving from the name Al-Khowarizmi, a mathematician who lived in the court of Mamun in Baghdad, in the early 9th century. The mathematician's name was Abu Ja'far Mohammed ibn Musa Al-khowarizmi. (Ah! We still find historical accounts to be a kind of explanation, don't we?) He left for posterity two books, one entitled *Hisab al-jabr wál-muqabala*, "The calculation of reduction and restoration," translated (and transliterated) three hundred years later into Latin (by Robert of Chester) as *Liber algebrae et almucabala*, a transliteration that bestowed the word *algebra* on the West. Al-Khowarizmi's other book was translated into Latin with the title *Liber Algorismi de numero Indorum*, and it is from this word that the word *algorism*, later *algorithm*, arose.[2] As the title suggests, it deals with the revolutionary idea of writing numbers with just 10 digits, from 0 to 9, and having at one end a 1's place, then a 10's place, a 100's place, and so forth – a notion that has truly changed the world.

Loosely speaking, an algorithm is an explicit and step-by-step explanation of how to perform a calculation. The classical ancient world developed a number of

[2] http://www-groups.dcs.st-and.ac.uk/history/Mathematicians/Al-Khwarizmi.html On Khwarazm: http://www.informatik.uni-trier.de/~ghasemzadeh/

significant algorithms, such as Euclid's algorithm for finding the greatest common divisor of two integers M and N. It is a process of continuing to divide one integer by another, starting with M and N, and holding on to the remainder, but each time dropping the dividend; once we find that the remainder is zero, the greatest common divisor is the last divisor in the operation. This remarkable path to the discovery of the greatest common divisor that bears Euclid's name has another important property that most people today take to be an essential part of being an *algorithm*: it is an operation that is guaranteed to work (that is, to be finished) in a finite amount of time (that is, a finite number of calculations) – though the specific value of the finite limit is likely to depend on the particular input that we choose to give to the algorithm.

In its origins, the algorithm was created in order to describe an operation too complex to explain in any other fashion, like Euclid's algorithm for greatest common divisor. But a revolutionary change took place when mathematicians and logicians came to consider the notion that *all* thinking about mathematics could be described as an algorithm, and that if this were so, Aristotle's attempt to classify and categorize sound inference could be greatly expanded to cover mathematics. These ideas were first discussed by Blaise Pascal and by Gottfried von Leibnitz, extended in the 19th century by Giuseppe Peano and Gottlob Frege, and developed in detail in the 1930s by Kurt Gödel, Alonzo Church, Alan Turing, and Emil Post.[3]

The second revolutionary change associated with the algorithm was the realization (which Pascal was already very aware of) that an algorithm was something that in many cases could be easily *embodied* in a physical object, in such a way that the steps of the algorithm corresponded to motions of physical parts. Man's control of physical nature had reached a point in the 1930s that it became possible for the first time to create a rapid, general, and practical implementation of algorithms, a device that we today call the computer. Alan Turing's effort to do just this during World War II is well known,[4] and was a major contributor to the defeat of Hitler's military codes during the war.

Although several technical ways were developed for expressing algorithms in all their explicit glory, it was an important result of the time that these different ways were at their heart all equivalent. Turing's formulation was the one that seemed the easiest to state in simple terms: given his imaginary (or abstract) *Turing machine*, any algorithm could be expressed as a sequence of 1's and 0's on a long piece of tape, and any sequence of 1's and 0's would be interpreted as an algorithm by the Turing machine.

---

[3] Berlinkski 2000 is an entertaining book on the history of the algorithm.
[4] It is also a story told in Neal Stephenson's *Cryptonomican* 1999.

If any algorithm could be represented as a finite sequence of 1's and 0's, then it could be easily (and naturally) assigned an integer: whatever integer those 1's and 0's represented in binary arithmetic. There was thus a natural order to the set of all algorithms: they could be lined up in ascending order, and every algorithm would have its place, determined in a straightforward manner.[5] It would then be possible to search for an algorithm just by stepping through each of the natural numbers, because each natural number *was*, in the sense we've just described, a representation of an algorithm. And if we're looking for an algorithm that accomplished a particular task (say, that generate the sentences of English), we would stop as soon as we found one that we could determine was capable of accomplishing the task in question (if we could find a satisfactory means of testing whether the algorithm did what we wanted it to do, of course).

Now, by the 1930s, logicians were already talking about generating logical languages with the formal mechanisms of algorithms. It was only the next step to try to apply the same ways of conceptualizing the problem to the task of understanding natural language as the product of an algorithm, as Zellig Harris and Noam Chomsky began to do in the late 1940s and into the 1950s.

Zellig Harris was developing the idea (among others) that the pursuit of the right grammar for a set of linguistic data was the most *compact* grammatical representation of the data. The focus on compactness naturally grew out of the way logicians were thinking about algorithms already at this point: as I pointed out, the same *idea* could be formalized (into a Turing machine program) in several ways, but two kinds of priority were to be given to the shortest formulation among a set of otherwise equivalent formulations. The priority derives, first, from the crucial observation that if we are searching for an algorithm by going through the universal inventory *in order*, going from lowest integer (0) to ever larger integers, we are naturally looking at all smaller programs before all longer programs: that is just a way of saying that a smaller integer is a shorter integer, after all.

But there was much more to the size, or compactness, of the algorithm, and during the 1950s this idea was pursued along a number of lines. From the linguistic point of view, pride of place goes to Chomsky's work in *The Logical Structure of Linguistic Theory* (1975 [1955]), in which the proposal is made that the goal of the designer of the linguistic theory is to build a theory in which the shortest grammar consistent with the data is the correct one. Chomsky noted,

[5] Different sequences of 1's and 0's (different Turing machine programs) might correspond conceptually to the same algorithm at a human level, of course.

> In careful descriptive work, we almost always find that one of the considerations involved in choosing among alternative analyses is the simplicity of the resulting grammar. If we can set up elements in such a way that very few rules need be given about their distribution, or that these rules are very similar to the rules for other elements, this fact certainly seems to be a valid support for the analysis in question. It seems reasonable, then, to inquire into the possibility of defining linguistic notions in the general theory partly in terms of such properties of grammar as simplicity. (p. 113-114)…In constructing a grammar, we try to set up elements having regular, similarly patterned, and easily statable distributions, and which are subject to similar variations under similar conditions; in other words, elements about which a good deal of generalization is possible and few special restrictions need be stated. It is interesting to note that any simplification along these lines is immediately reflected in the length of the grammar. (117) …It is tempting, then, to consider the possibility of devising a notational system which converts considerations of simplicity into considerations of length….(More generally, simplicity might be determined as a weighted function of the number of symbols, the weighting devised so as to favor reductions in certain parts of the grammar.) (117) It is important to recognize that we are not interested in reduction of the length of grammars for its own sake. Our aim is rather to permit just those reductions in length which reflect real simplicity, that is, which will turn simpler grammars (in some partially understood, presystematic sense of this notion) into shorter grammars. 118.

At the same time, other developments were taking place, in both the U.S. and the Soviet Union. In the Soviet Union, the distinguished mathematician Andrej Kolmogorov (1903-1987) was developing a notion by which any algorithm could be assigned an *apriori* probability. You will recall that to assign probabilities to a set of distinct items, the probabilities must add up to 1.0, and this obviously requires some care when considering an infinite set, such as the set of all algorithms. He proposed that the probability is directly and simply based on the *length* of the shortest implementation of the algorithm, and it is not difficult to see that if a binary number of length N is assigned a probability equal to $1/2^{2N}$, then these probabilities will sum to 1, and shorter programs will be assigned a higher *apriori* probability: in fact, two programs whose lengths differ by *d* will have probabilities whose ratios are 2^{2d}.[6] Very similar ideas were being developed at the

[6] I leave aside some technical points that are irrelevant to the overall ideas involved, such as the fact that we may decide to assign a higher probability than this formula suggests, while at the same time allowing only certain strings of binary digits to represent legitimate algorithms.

same time in the United States, first by Ray Solomonoff[7] and a bit later by Gregory Chaitin (Li and Vitányi 1997).

4. It's fine to have an apriori rating for a grammar, but how well does it deal with the data?

In the development we have considered up to this point, the focus has been on the grammar, and developing an *apriori* (or "prior") evaluation on the goodness of the grammar as such. But that's only half the story in selecting a grammar: we also need to know how well the grammar jibes with the data in question. Chomsky's *LSLT* touches on the question of disagreement between grammar and data (in particular, in his chapter 5 on grammaticalness), but ultimately has little to say about the problem. And the problem is this: would we be willing to accept a short grammar for our data even if it did not generate *all* of the data? If so, how much data are we willing to abandon accounting for with each shortening of the grammar? What is the fundamental nature of the trade-off between conciseness of grammar and fit to the data? Generative grammar had nothing to say to this question.

Ray Solomonoff was very directly working on just this problem in the mid 1950s, years later posing the problem in a way that sounds to a linguist's ears just like the problem of grammar induction: given a finite amount of data, how do we decide what is the best description of the data? In Solomonoff's words:

> On reading Chomsky's "Three Models for the Description of Language" (Cho 56), I found his rules for generating sentences to be very similar to the techniques I had been using in the 1957 paper to create new abstractions from old, but his grammars were organized better, easier to understand, and easier to generalize. It was immediately clear that his formal languages were ideal for induction. Furthermore, they would give a kind of induction that was considerably different from techniques used in statistics up to that time. The kinds of regularities it could recognize would be entirely new. [Solomonoff is undoubtedly referring to Markov models here – JAG]
>
> At the time of Chomsky's paper, I was trying to find a satisfactory utility evaluation function for my own system. I continued working on this with no great success until 1958, when I decided to look at Chomsky's paper more closely. It was easy for me to understand and build upon. In a short time, I devised a fast left to right parser for context free languages and an extremely fast matrix parser for context sensitive languages. It took advantage of special 32 bit parallel processing instructions that most computers have.

[7] See Solomonoff 1995 for a very readable account.

My main interest, however, was learning. I was trying to find an algorithm for the discovery of the "best" grammar for a given set of acceptable sentences. One of the things sought for: Given a set of positive cases of acceptable sentences and several grammars, any of which is able to generate all of the sentences - what goodness of fit criterion should be used? It is clear that the "Ad-hoc grammar", that lists all of the sentences in the corpus, fits perfectly. The "promiscuous grammar" that accepts any conceivable sentence, also fits perfectly. The first grammar has a long description, the second has a short description. It seemed that some grammar half way between these, was "correct" - but what criterion should be used?

There are other modes of learning in which the "goodness of fit" criterion is clearer.

One such learning environment involves a "teacher", who is able to tell the "learner" if a proposed sentence is within the language or not. Another training environment gives negative as well as positive examples of sentences. Neither of these training environments are easy to obtain in the real world. The "positive cases only, with a few errors" environment is, by far, most widely available.

The real breakthrough came with my invention of probabilistic languages and their associated grammars. In a deterministic (non-probabilistic) language, a string is either an acceptable sentence or it is not an acceptable sentence. Taking a clue from Korzybski - we note that in the real world, we usually don't know for sure whether anything is true or false -but we can assign probabilities. Thus a probabilistic language assigns a probability value to every possible string. In a "normalized" language, the total probability of all strings is one.

It is easy to give examples of probabilistic grammars: any context free or context sensitive generative grammar can be written as a set of rewrite rules with two or more choices for each rewrite. If we assign probabilities to each of the choices, we have a probabilistic grammar.

The way probabilistic grammars define a solution to the "positive examples only" induction problem:

Each possible non-probabilistic grammar is assigned an a priori probability, by using a simple probabilistic grammar to generate non-probabilistic grammars.

Each non-probabilistic grammar that could have created the data set can be changed to a probabilistic grammar by giving it probabilities for each of its

choices. For the particular data set of interest, we adjust these probabilities so the probability that the grammar will create that data set is maximum.

This idea of a probabilistic grammar has become a standard concept in the field of computational linguistics; one cannot work in speech recognition, and it is *nearly* impossible to work in syntactic parsing, without using this notion directly. It employs the notion of probability in a thorough-goingly formal fashion; it shares little or nothing with the urge to view language probabilistically because of some perceived fuzziness in linguistic categories or rules.

Summarizing so far: formal analysis provides a new kind of explanation, and probability theory is a method to test two aspects of a given analysis – it can test, using Solomonoff's idea, the goodness of fit between the formal grammar and the data, and secondly, the preference for a concise grammar can be expressed as a probability, a *prior* probability based on the grammar's formal length. Given the nature of probability theory, it is possible to put these two probabilities together, and give a single evaluation for how well a grammar evaluates a set of data, that is the probability of the grammar given the data, by combining (multiplicatively) the prior probability of the grammar, on the one hand, and the probability that the grammar assigns to the data, on the other.[8]

If we knew nothing about the history of linguistics and had heard the story up to here, I think we'd expect that the application of generative models of grammar to computational ends would be (or would have been) a piece of cake – nothing more natural. In fact, the history was not like that at all, and in the last 15 years, as there has been more and more computational work on natural language, it has not been generative grammars that have naturally been applied to this work. Why should this be? Is it a sign that something's wrong somewhere, and if it is, what's wrong?

5. From algorithm to generative grammar, from generative grammar to algorithm

The crooked path that we have followed so far has been intended to illustrate the following idea: the development of generative grammar is a step that took place in a broader intellectual development that was tightly rooted in conceptual developments in the foundations of logic, mathematics, and computation from the 1930s through the 1960s, and the belief that a concise, algorithmic account of a linguistic corpus is a valid form of explanation of the data is of a piece with this intellectual movement.

[8] This evaluation is not a probability unless we divide this product by the probability of the data, however.

Two things happened in linguistics in the 1960s that bear on this observation. The *first* was that Chomsky emphasized considerably more strongly the human and cognitive base for generative grammar than he had in the 1950s (and eventually, in the late 1970s, all connection between grammar length and grammar preference was abandoned, with the introduction of the remarkably simplistic vision of principles and parameters grammars, and essentially a total abandonment, with little comment or fanfare, of the central substantive notion of generative grammar, that of an evaluation metric); the *second* was that it became possible to start implementing grammars computationally.

Of course, grammars had been implemented computationally before; Chomsky's first employment as a linguist had been working in Victor Yngve's computational group at MIT, developing some aspects of generative grammar computationally.[9] And a number of research groups were implementing grammatical models by now, including Zellig Harris's group at Penn, and Sydney Lamb's group at Yale. [refs] MITRE at MIT, Kuno and Oettinger at Harvard. We could point to the early meeting in June 1952 of the MIT Conference on Mechanical Translation, organized by Yehoshua Bar-Hillel, as an indicator of the beginning of organized work on machine translation (MT), and to the founding of the "Association for Machine Translation and Computational Linguistics" ten years later, in June of 1962, as the beginning of a serious movement in computational linguistics in this country.

But with a small number of notable exceptions (the work on Lexical Functional Grammar and the work on HPSG being the prime examples, but there are dozens of others), mainstream American linguistics has remained aloof from computational applications and implementations. It would be a long paper indeed – it would be a book – that surveyed the various ways in which computational linguistics borrowed from mainstream linguistics, and the ways in which mainstream linguistics borrowed notions developed in computational communities, and this is not that paper.[10] We will limit ourselves to just one question, which is:

---

[9] See, for example, Huck and Goldsmith 1995; see also Yngve 1982.

[10] A question was asked at the meeting after this paper was presented: does not the message offered here risk making linguistics more dependent on the vagaries of the current computer metaphor, at a time when we would prefer to have our notions be motivated by strictly linguistic concerns? The answer to this (quite reasonable) concern, I suggested, is that linguistics already is heavily dependent on the vagaries of current computational metaphors, notations, and assumptions – probably far more than most linguists are aware. The role played by feature inheritance and unification in both LFG and HPSG are good examples of this

5.1 Why don't we really try to minimize our grammar lengths? So why don't we really try to minimize the length of the grammars we write? Oddly enough, I think this brings us to the heart of the question, and I'll give a simple answer to this question. The answer is: it doesn't do any good to try to minimize the length of your grammar unless you add the restriction that your grammar is responsible for dealing with all of the data, where "all of the data" is established in some fashion antecedently to the analysis – typically by collecting a corpus of data ahead of time. If you don't set such a condition, then it is truly inevitable that the analysis will be given free rein to pick and choose the data that it analyses best, and that is tantamount to giving the analysis permission to make itself very short.

The demand to make the grammar compact makes sense, and works, only if we set a condition that the grammar must deal with all of the data, and that includes the data that seems somehow "wrong," and it also includes all of the data that lies inbetween the data that is handled very well and the data that seems really wrong. It is sometimes surprising just how much data lies in that inbetween area; in many cases, it is the bulk of the data. If we are interested in morphology, then we need a morphology of the data that analyzes every word in the corpus; if syntax, then a grammar that includes every node expansion and so forth.

It's only in this way that grammars can be compared, and I'm emphasizing here two distinct points: first, if they don't make the effort to describe the same data, then two grammars are not comparable, and second, if we don't establish grammars of a language (or corpus) that describe all of the data in the corpus, then we haven't pushed the grammar, we haven't put it to the test – where the test consists not just of dealing with the cases that come out nicely, but all of the rest as well.

The problem of dealing with all of the data is essentially the problem of scaling, which is to say, extending an analysis from a toy set of data to a realistically large set. This challenge is one of the central concerns of work in computational linguistics, and just about all of the work on applications involving computational implementations. Scaling up to real data (from toy data) presents all sorts of challenges, and the heart of the problem lies in the fact that you just don't know what the hardest challenges will be in a particular case until you actually undertake it. Linguists have mixed feelings about this challenge: linguists dealing with individual languages that have not been well-studied often view this challenge very positively, recognizing that every morpheme and every word is a

---

phenomenon: the exploitation by linguists for their own purposes of ideas developed in the computational community.

value unto itself in the language. Mainstream theoretical linguists do not always share this perspective, perhaps due to the feeling that value is not uniformly distributed across data, and the challenge to the linguist is to find those data of great value: these are the data which can be shown to shed bright light on theoretical divides, serving as decisive data in crucial experiments setting one theory against another.

6 Two applications

It might be a good idea to descend from these abstract altitudes and discuss a couple of application areas, where algorithmic interpretation of linguistic problems have been applied computationally. I'll discuss two briefly: first, speech recognition, and then the area that I have been working on, unsupervised learning of morphology.

Speech recognition involves the assignment of a correct grammatical parse to a sentence. But in particular, speech recognition requires what's called in the biz a *language model* that determines, from the point of view of what is known about the *language*, what is the most likely word-interpretation of a chunk in the sound stream. Computational speech recognition, just like human speech recognition, is an interplay of two things: paying close attention to the sound stream and using your knowledge of what's been said and your knowledge of how the language works to decide what the most likely interpretation of the sounds. "Your knowledge of how the language works": that phrase sounds remarkably like words we linguists use to describe grammar. So the question naturally arises, can speech recognition systems use linguists' grammars to serve the function of a language model in a speech recognition system?

To date, the answer has largely been No, and this may come as a surprise to linguists. There are three major reasons for this negative answer: 1. a probabilistic model is necessary, in order to integrate knowledge of language with perception of sounds coming in; 2. knowledge of words that have been used is extremely important, and largely more important than syntactic structure; 3. current grammatical models are very poor at dealing with a question like: if a sentence begins with the following string of words, figure out what their syntactic structure is and the likely candidate categories and words to immediately follow. Syntactic theory largely works with entire sentences, and speech recognition doesn't have that luxury.

I should say a bit more about the notion of a probabilistic model. Probabilistic models are versions of familiar formal grammars to which some additional formal devices have been added which permit the human who is evaluating them to determine quantitatively how well they fit the data. It's perhaps worth making explicit that (despite rumors to the contrary) probabilistic models are not different in formal ways from non-probabilistic models, except that they may be a bit more

formal than formal (since there are some mathematical conditions that they must respect). The critical property that probabilistic models have is that in assigning a probability to each analysis that is consistent with a set of data (in the case at hand, we're talking about assigning a syntactic structure on the basis of the words that we believe we have heard so far), it's possible to merge or integrate those probabilities with the probabilities which the *sound* part of the speech recognizer is simultaneously developing. To repeat, a speech recognition system consists of two parts, a sound model which attempts to determine what linguistic sounds most reasonably fit the physical sounds which are input to the device (crudely speaking, what phones most likely correspond to the rich physical input) and a language model, which can determine what is a reasonable or likely continuation of the words, with their structure, that have been said up to this point. The central point is that if both the speech and the language model are probabilistic, then they speak the same conceptual language, and it is easy to integrate the information that they are computing: it is most likely that what the speaker is actually saying is that which is *jointly*, or *at the same time* the most likely word, given the sounds and given the expectations set up by the words and grammatical construction up to that point. A probabilistic model is well suited for judging what the relative ranking is of various alternative analyses, given some particular set of data.[11]

One of the most striking things about the models used in most speech recognition systems is how little they exploit what is known about phonetics and phonology. The standard models take into account the notion that speech is a realization of a sequence of phonemes, but little else: little or no knowledge of dialect variation, of prosodics, of intonation, of features or feature structure, of underspecification, or anything else that phonologists and phoneticians worry about and, we may hope, have learned a good deal about. Some phonologists have taken this observation to heart, but quite few, and the fact of the matter is that if a more sophisticated phonological approach were to bring real improvements to speech recognition, they would be instantly adopted by the speech community, which cares very much about performance. Phoneticians and phonologists should explore the value that their experience and their theories can bring to this area.

The second application that I would like to say a bit about is a system called *Linguistica* that I have been working on for several years. It is a system whose

[11] If this sounds vaguely like optimality theory, and the perspective that a theory should be called upon to choose the top candidate of a ranked set, rather than to establish a fence between those representations that are legitimate and those that are not, it is no coincidence; optimality theory can certainly be viewed as a computational system for approximating the calculation of probabilities, if a bound on the number of violations a constraint can have is established.

goal is to automatically analyze the morphological structure of a language; I reported on its early design in a paper to the Chicago Linguistic Society in 2000 (see also Goldsmith 2001). I have to think twice when asking myself whether it is in fact an *application*, and whether it is an application *of linguistic theory*. It is a stand-alone system which accepts as input linguistic data, either in phonetic or in orthographic form, and performs an analysis which could quite naturally be of use to someone other than its designer, so that makes it an application, I would say. The publisher of a multi-lingual electronic or on-line encyclopedia could use such a system to automatically create a morphologically-aware index, in the following sense. If we were designed an encyclopedia like Microsoft's Encarta in a variety of world languages, we would want the user to be able to enter any inflected form of a word, and to be able to access paragraphs containing alternate word-forms in the same lexeme as the one the user had typed in. This is a relatively well-understood problem in English, but it is a challenge in a language for which computational morphologies are not easily available. Such a company could easy perform a morphological analysis of the entire corpus in a few minutes, and achieve the goal in short order.

There are many challenges in building such a system, but perhaps the most striking one is the globality of the problem. Perhaps that is not the best word to describe the challenge, but what I have in mind is this. In contemporary linguistic theory, it is a commonplace that the linguist who gets ready to tackle a problem has to define well what he or she will deal with, and in the same moment a whole host of other problems are left aside. There is nothing wrong with this, to be sure, and linguistic theory could not advance if this were not the norm. But the development of an application is often forced to deal with a range of questions that current theories have little interest in. The theories may have no interest in it, but the application requires their solution if it is to move forward.

I have been faced specifically with this situation in the development of automatic morphological analysis, in the following way. Morphological analysis is a significant and sophisticated subfield of mainstream linguistic theory, with a host of theoretical questions ranging from morphology's interaction with semantics, to its interaction with syntax, to its interaction with phonology. But the down-to-earth task of determining how many morphemes there are in a word, where they begin and where they end, is question that linguistic theory frankly does not take very seriously: we may expect our students to figure out how to do it in an introduction to linguistics, but no theoretical question in mainstream morphology I know of is answered by finding a better way to discover the morphological parsing of a word. But: if finding a better way to discover the morphological parsing of a word is what it takes to turn linguistic theory into an application, then it is linguists who should take up the challenge: there is nobody better equipped than us to perform the job!

In the end, of course, I think that there is a great deal to be learned by developing systems that stay close to the data, as all applications do.

7 In conclusion

I will conclude by putting together again some of the points we have been discussing. First of all, that there are deep connections between the algorithm and the rise of generative grammar; second, that these connections suggest a deep connection between formalist accounts of language and a thorough-goingly empiricist account of natural language, rather than a rationalist point of view; third, viewed in this way it becomes necessary to consider not only a formal treatment of the internal structure of the grammar but also a formal (which is to say, quantitative) treatment of the relationship of the grammar to bodies of evidence; and fourth, bodies of evidence can play this role most legitimately when they are established independently of the grammars which are conceptually responsible for explaining them.

Such a style of formal linguistic analysis has a better chance to undergo a transformation to an application – if you should believe that to be an important goal. (I do.)

On the other hand, algorithmic explanation is only one of several kinds of explanation; others that continue to play an important role in linguistics include *historical, sociological* and *psychological* explanation. How do we make choices among these different types, or styles, of explanation, and how do we evaluate their (relative) success? Ultimately in three ways, it seems: first, the degree of insight that they offer the scientist; second, their ability to connect hitherto unrelated fields of inquiry; and third, their ability to give rise to useful applications. These are the criteria by which any scientific (and many non-scientific) approaches are evaluated. In light of this, the take-home message (as I see it) is that formal linguistics should be looking for more applications, and that more and better applications will be possible as formal accounts develop probabilistic models that offer an account of a wide body of data.

References

Chomsky, N. (1975 [1955]). *The Local Structure of Linguistic Theory*. New York, Plenum.

Bar-Hillel, Y. (1960). "The present status of automatic translation of languages." *Advances in Computers* **1**: 91-163.

Berlinski, D. (2000). *The advent of the algorithm : the idea that rules the world.* New York, Harcourt.

Goldsmith, J. (2000). **Linguistica**: An Automatic Morphological Analyzer. *The Proceedings from the Main Session of the Chicago Linguistic Society's Thirty-sixth Meeting. Volume 36-1*. A. a. J. B. Okrent. Chicago, Chicago Linguistics Society**:** 125-139.

Goldsmith, J. (2001). "Unsupervised Learning of the Morphology of a Natural Language." *Computational Linguistics* **27**(2): 153-198.

Huck, G. J. and J. A. Goldsmith (1995). *Ideology and linguistic theory : Noam Chomsky and the deep structure debates.* London ; New York, Routledge.

Li, M. and P. M. B. Vitányi (1997). *An introduction to Kolmogorov complexity and its applications*. New York, Springer.

Solomonoff, R. (1995). The discovery of algorithmic probability: a guide for the programming of true creativity.

Stephenson, N. (1999). *Cryptonomicon*. New York, Avon Press.

Yngve, V. (1982). Our double anniversary. http://acl.ldc.upenn.edu/P/P82/P82-1018.pdf