ACL Lifetime Achievement Award

A Life of Language

Martin Kay*
Stanford University

Introduction

This is a truly overwhelming experience. Since I am not a cat, I expect to have one lifetime in which to achieve anything, and one award for it at the most. So this is it, and I am honored and humbled and, of course, delighted. It is wonderful to be among so many of the friends who have enriched my life in so many ways. Thank you for permitting me to feel, for some short moments, that one or two of the things I have done may be allowed to count as accomplishments.

There is only one thing that gave me pause for a moment when I heard about the award, and that was its title. But my concern was soon put to rest when I checked up on the previous three recipients and found that, as far as I could tell, they were doing quite well.

If you will forgive me, I will tell you a little about this lifetime, such as it has been. After all, as I have said, one has but one chance for such gross indulgence.

The ambiguities that are a linguist's constant companion were with me from the start for, if I say I was born in London, then do I mean that the place where I was born was part of London at the time of my birth, or that it is part of London now? In fact, only the second proposition is true. I was born in 1935 in Edgeware, which was in the county of Middlesex then, but was later absorbed by London for reasons that I believe to be unrelated.

The first event that lodged in my memory was the declaration of war in 1939. I had no idea what it meant, but everyone took it so seriously that it made some kind of impression on me even at the age of four. London was bombed a lot, and Edgeware a little. My mother was afraid of the bombing, but I found it fun. My father was never afraid of anything. He was an inspector of schools and was charged with evacuating children from London. I was sent to a variety of places but invariably returned to London after a few weeks because my mother could not stand to be away from the excitement of the capital for more than a few weeks, even if it did mean being bombed. We went to Dorchester, and Devizes, in the west country, and I encountered new kinds of vowels and discovered that syllables could end in an "r". I started mimicking people and learning to associate the way people talked with where they came from. We went to Leeds, in the north country, and I learned how diphthongs could become monophthongs and how to tell whether a person came from Leeds or Bradford, despite the fact that these cities are barely fifteen miles apart. People told me I was destined to become a linguist. I did not know what a linguist was, and it turns out, in retrospect, neither did they.

* Linguistics Department, Margaret Jacks Hall, Stanford CA 94305-2150. This paper is the text of the talk given on receipt of the ACL's Lifetime Achievement Award in 2005.

One of the places I was sent to avoid the bombs was Llambadarn, a small village in southern Wales where only Welsh was spoken. The idea that there could be a place where people did not speak English had not occurred to me before, and it fascinated me. One of the things I still regret about the experience is that my mother could not stand it for more than three weeks. Had she been able to, I might have learned Welsh.

Why I was called "Martin," I do not know. The plan for my life that was beginning to take shape seems to have attracted people called "Henry" in the past. For example, there was Henry Lee Smith, who had a radio show in the United States. After hearing somebody talk for 30 seconds, he told them where they came from, where they went to school, where they had moved to after that, and so forth. He would give a history of the person based on how they spoke. Something like that had already become my "party piece" when I was six or seven. Then there was Henry Sweet, who did similar kinds of things in Britain, and another well-known character called Henry Higgins, who was actually based on Henry Sweet, and so should not, I suppose, really count.

In 1947, I was sent to one of those rather exclusive places the British like to call "public schools" to get an education. Soon after, since I wanted to learn languages, I went to Tours to learn French and Baden-Baden to learn German. Then I went into the army to defend Britain. And, against the kinds of things that I am equipped to defend a country against, I defended it magnificently.

After that, I went to Cambridge to get more education and, as they say in Britain, to *read* modern and medieval languages. Reading a subject in Cambridge is approximately like majoring in it in America, except that there is nothing *but* the major. You spend all your time on what you are reading; there is nothing else. Modern and medieval languages constituted about the closest thing I could find to what I was looking for, but somewhere in the back of my mind, I knew that what I wanted to do was linguistics, even though all I knew about it was what adults had said to me when I was a child. It turns out that it did exist. In fact, it existed in Michael Halliday's department, within walking distance of where I lived in London. Whether it would have been better to go there, or whether the experience of going to Cambridge was worth it in its own right, I do not know. But, instead of doing what Henry Sweet and all those other Henrys' did, I wound up reading *Le Rouge et le Noir, La Divina Comedia, Faust*, the *Chanson de Roland, Parzifal* (with a "z") and other literary monuments. But I found myself saying all the time, "What about language? Doesn't all this come from ordinary language somehow? Isn't the language that ordinary people speak a wonderful thing that deserves to be studied in its own right?"

I remember asking my French supervisor this question once, and I have to admit that the answer he gave me, wrong though it was, has remained in my memory ever since because I simply could not find any refutation for it at the time. What he said was, "If you want to study French, don't you want to study it as used by those who use it best?" In other words, would I not want to study the French of the literary giants who created great works of art in that language?

Lord Adrian, the master of my college, and also vice chancellor of the university, asked me to act as his interpreter when giving an honorary degree to Signor Gronchi, president of Italy. The president had a cold and insisted on wearing his doctoral gown over his raincoat. But he took me for a beer at the Blue Boar afterwards, and I forgave him everything.

When I was at Cambridge, it turned out that the undergraduate engineering society ordered a film from London, and the people at the film library put the wrong film in the envelope. Instead of something on steam turbines, or whatever, they got Kurasawa's *Roshomon*. I had for many years wanted to see this film. I don't know what

it was that excited me about it. Maybe it had something to do with language, again. Maybe the idea of four people telling essentially the same story, and it coming out differently as different words were woven around it—maybe that was what fascinated me. As you probably know, it is the story of a murder, told by four witnesses one after the other. One of the accounts is that of the murdered man himself, told through a medium.

In order to see this movie, which had always eluded me up until that point, I had to join the undergraduate engineering society. The literature that I received in the mail after joining told me, among many other things, that it offered a prize of 25 pounds every year for the best paper delivered by an undergraduate. I could not avoid asking myself what a medieval linguist would have to do to secure such a prize. A serious paper would obviously stand no chance. What I had to do was find a subject that would cause people to laugh, because there was no intersection in the serious world between anything I knew and what were serious matters to them.

I decided I would offer a paper on a translating machine, an idea which, surely, nobody could take seriously. I gave the paper and stimulated some laughter. However, sitting in the audience were Margaret Masterman and Frederick Parker Rhodes. Margaret Masterman was the director of the Cambridge Language Research Unit, and Frederick Parker Rhodes was a senior researcher there. They had American money to work on machine translation. Margaret Masterman's many outstanding properties did not include a robust sense of humor. She found my paper impertinent and insisted that I visit the language unit to learn the truth about machine translation. To cut a long story short, the following year, I became a very junior member of the Cambridge Language Research Unit, having foresworn my evil ways and shown that I could speak Italian and use a soldering iron.

I worked for several years with Margaret Masterman. She was the wife of Richard Brathwaite, the Knightsbridge Professor of Ethics whose major work, however, was on the philosophy of science. Margaret Masterman was a student of Wittgenstein, a fact that she never let you forget, because it gave authority to even the most outrageous things she might choose to say. She did pioneering work on semantic nets with R. H. Richens and had a theory of translation based on thesauruses and whose principal formal tool was lattice theory. She was a member of a shadowy society called the "Epiphany Philosophers," who took it as their goal to show that Christianity and science were not only compatible but that they supported one another. My father confused the Epiphany Philosophers with the Apostles, a secret society founded in the early nineteenth century and intended to have as its members the 12 brightest undergraduates in the university. My father did not know this part of the history. What he did know was that it was based in my college—Trinity—and that its recent members had included the infamous Cambridge Four—Kim Philby, Donald Maclean, Guy Burgess, Anthony Blunt—all engaged in spying for the Soviet Union. A patriot who was concerned for my welfare, as parents are wont to be, my father occasionally expressed the hope that I was devoting my linguistic skills to worthy pursuits and that MI6 would not be coming to visit.

Margaret Masterman was one of the cofounders of Lucy Cavendish College, a most unusual institution that admits as undergraduates women who have been away as mothers and who want to come back into mainstream academic life. It is a remarkable place, and she was a remarkable woman.

The Cambridge Language Research Unit was small, but it had a number of illustrious alumni. Michael Halliday had spent some time there, mainly as an expert on Chinese. The late Roger Needham started his career there, as did his wife, Karen

Spark Jones, the previous recipient of this award. Yorick Wilks, surely well known to this audience, also started his career there.

After two years at the Cambridge Language Research Unit, I seized upon an opportunity that came up to visit the Rand Corporation in California for six months. This was a trip to the United States that I have been extending ever since. There, I worked for another remarkable person, David G. Hays, to whom we owe, among other things, the Association for Computational Linguistics, the International Conferences on Computational Linguistics, and the very name "Computational Linguistics." At the time, the Rand Corporation was working on Russian machine translation and was ahead of its time in that it had constructed a million-word dependency tree bank of Russian already in 1962 when such things were somewhat less fashionable than they are today.

While I was at Rand, I took over from Hays the organization of a set of seminars to which people interested in computers and language came every week. Several of those people told me later that the meetings had played an important role in determining the later course of their lives. Also, about the same time, I started teaching computational linguistics at UCLA.

At Rand, I had the great privilege of rubbing shoulders with a wonderfully rich variety of people who either worked there or who came on extended visits. Some names that come immediately to mind are those of George Danzig, inventor of the Simplex Method for Linear Programming; Norman Dalkey, one of the originators of the Delphi method; Newell, Simon, and Shaw, who could be said to have invented Artificial Intelligence; Albert Wohlsetter, who once called the Secretary of State from my house to clarify a question that arose over dinner; Keith Uncapher, founder of the Information Sciences Institute at the University of California; Richard Bellman, who created dynamic programming; Daniel Ellsberg of *Pentagon Papers* fame; Paul Baran, who first proposed packet switching for computer networks; and Herman Kahn, who founded the Institute for the Future and who played war games where the score was kept in megadeaths. I would have it known that I had been gone from Rand for 10 years when Donald Rumsfeld became its chairman in 1981 and 20 years before Condoleezza Rice joined its board of trustees.

For some years, the linguistics project at Rand had an internship program that included a number of people who are still prominent in our field, and for very good reason. They include my old friends and long-time collaborators, Ron Kaplan and Lauri Karttunen.

In 1962, the Association for Machine Translation and Computational Linguistics (AMTCL) came into being. A couple of years later, the first International Conference on Computational Linguistics was held in New York. These conferences have now come to be called "Coling," so named by a Swede, who therefore pronounced it "Cooling," which is also how a Swede pronounces "Koling," the name of Albert Engström's much-loved cartoon character, a vagabond who made sage remarks about the world between swigs at a bottle of wine. This name was associated with the conferences organized every other year by the International Committee on Computational Linguistics (ICCL) by the late Hans Karlgren. I became chair of the ICCL in 1984 and have been there ever since.

The establishment of the AMTCL and ICCL foreshadowed an extremely momentous event that came about because the people involved also had to do with a document known as the *ALPAC Report*. This was one of the most well known, if least read, documents ever produced concerning our field. Its full title was "Languages and Machines: computers in translation and linguistics." It was called the *ALPAC Report*

(ALPAC) after its authors, the "Automatic Language Processing Advisory Committee," established by the U.S. National Academy of Sciences. It was a mere quarter of an inch thick, with a black cover altogether appropriate to its contents. The remit of the committee was much narrower than most people usually appreciate. The committee was supposed to tell the government how successful, how useful, how worthwhile the research that was being devoted to machine translation with government money was going to be. They were to concern themselves with the potential benefits for the government, not for anybody else. But, since the money for the work came from the government, a negative report could have dire consequences for everybody, and it did.

ALPAC discussed a number of interesting questions, but the essential conclusion consisted of two points. First, the government did not really need machine translation and second, even if it did, the lines of research that were being pursued had little chance of giving it to them. So the work should stop, and stop it did. A second recommendation of the committee almost got lost. It was that there should be a new focus of attention on the scientific questions that might provide a solid foundation for later engineering enterprises like machine translation. It was with the clear aim of responding to this recommendation that this association and these conferences came into being.

Now anybody who competes for research grants knows that while substance and competence play a significant role, the most important thing to have is a name, and we did not have one for the exciting new scientific enterprise we were about to engage upon. To be sure, the association and the committee antedated that report, but we had inside information, and we were ready. I use the word "we" loosely. I was precocious, but very junior, so that my role was no more than that of a fly on the wall. However, I was indeed present at a meeting in the office of David Hays at Rand when the name "computational linguistics" was settled upon. I remember who was there, but in the interest of avoiding embarrassment, I will abstain from mentioning their names. As I recall, four proposals were put forward, namely:

1. Computational Linguistics
2. Mechanolinguistics
3. Automatic Language Data Processing
4. Natural Language Processing

Strong arguments were put forward against the latter two because it was felt that they did not sufficiently stress the scientific nature of the proposed enterprise. The term "Natural Language Processing" is now very popular, and if you look at the proceedings of this conference, you may well wonder whether the question of what we call ourselves and our association should not be revisited. But I would argue against this. Indeed, let me briefly do so.

Computational linguistics is not natural language processing. Computational linguistics is trying to do what linguists do in a computational manner, not trying to process texts, by whatever methods, for practical purposes. Natural language processing, on the other hand, is motivated by engineering concerns. I suspect that nobody would care about building probabilistic models of language unless it was thought that they would serve some practical end. There is nothing unworthy in such an enterprise. But ALPAC's conclusions are as true today as they were in the 1960s—good

engineering requires good science. If one's view of language is that it is a probability distribution over strings of letter or sounds, one turns one's back on the scientific achievements of the ages and foreswears the opportunity that computers offer to carry that enterprise forward.

Statistical approaches to the processing of unannotated text bring up the thorny philosophical question of whether the necessary properties of language are, in fact, emergent properties of text. Could it be that at least some of the facts that one needs to know about a text are not anywhere in it? There is one sense in which the answer has to be "no" for, if they are not in the text, then they are not facts about the particular text, but about texts in general, or about some class of texts to which the given one belongs. But an extreme case that might dispose one rather to answer "yes" would be one in which the "text" simply consisted of the library call number of another text that contained the real information. Someone who knows enough to be able to find what the call number leads to can find what the text is really about, but it is not spelled out in the original document. When people say that language is *situated*, they mean that examples of language use always have some of this latter quality. They depend for their understanding on outside references that their receivers must be in a position to resolve. I will give a concrete example in a moment.

Part of the problem we are confronting comes from what the famous Swiss linguist Ferdinand de Saussure called *l'arbitraire du signe*—the arbitrariness of signs. The relationship between a word, a text, or any linguistic item, and its meaning is entirely arbitrary. It therefore does not matter how long you look at a text; you will never discover what it means unless you have some kind of inside information. It may very well be that the relationship between a sentence and its structure is also arbitrary, though here the situation is less clear.

This is hardly surprising. All it amounts to is that you cannot understand a text unless you know the language. You can, however, learn a lot about the translation relation from a text and its translation. If that were not the case, Jean Franois Chapolion would not have been able to decipher the Egyptian hieroglyphs on the basis of a stone that contained a translation of hieroglyphic Egyptian into Greek. It seems to follow, therefore, that meaning has little that is essentially to do with translation. Since you cannot get the meaning from the string, but you can find out about translation from two strings, then presumably the meaning is not directly involved. Let me argue against this position by giving a counterexample. There is nothing unusual about this counterexample. Examples of this kind are not unusual. At least one can be found in almost any paragraph-sized example of everyday language.

I was sitting in a train in Montpelier, and an old lady got in and said, "Does this train go to Perpignon?" The person she was addressing said, "No, it stops in Béziers." What could be simpler than that? Let's try and translate it into German. "Fährt dieser Zug nach Perpignon?" No trouble with that as far as I can see! "No, it stops in Béziers"—"Nein, er endet in Béziers." So, you should imagine a railway line that runs from Montpelier to Perpignon by way of Béziers. If a train were to end its journey in Béziers, it would never reach Perpignon.

But suppose the situation on the ground were different. Suppose that, after leaving Montpelier, there were a fork in the line, with one branch going to Perpignon and the other to a place called Findeligne by way of Béziers. Suppose, furthermore, that it is well known that all trains end their journey in either Findeligne or Perpignon. The interaction with the lady fits the new situation just as well as the old one. Since the train stops in Béziers, it must be a Findeligne train, and Perpignon is on the other branch. The German translation, however, must now be different. We can no longer trans-

late "No, it stops in Béziers" as "Nein, er endet in Béziers" because "endet" means "stop" only in the sense of completing the journey. We now need to translate "stop" in the sense of "come to a brief stop to allow passengers to get on and off," and for this, the appropriate word is *hält*. It therefore seems that, in order to be able to translate this passage correctly, one needs intimate knowledge of the geography of Provence and the schedules of the trains that run there.

Another problem with attempting to learn language from text is Zipf's Law. Zipf's Law says that a small number of phenomena—letters, words, rules, whatever—occur with very great frequency, whereas a very large number occur very rarely. Zipf's Law provides encouragement to people just starting to work on language because it means that almost anything you try to do with language works wonderfully at first. You do something on 100 common words with 20 rules designed for unremarkable situations and it works wonderfully. The trouble is that new phenomena that you had not thought of continue to appear indefinitely, but with steadily decreasing frequency. It is true that a textual example does not have to exemplify a new phenomenon in order to be interesting, because, as well as new phenomena, we are often interested in the frequencies of occurrence of old ones. But as a method of learning about different kinds of phenomena, it is subject to a crippling law of diminishing returns. Fortunately, there is an alternative, which is to talk to people who speak the language and who know what the phenomena are. This is what linguists do.

Another question is: Do the models that we build actually respect the fact that language is in accordance with Zipf's Law? If a probability distribution is established over a set of characters, and random text is generated in accordance with that distribution, the expected lengths of "words" will be determined by the probability of the space character, and the distribution of words will be approximately in accordance with Zipf's Law. If, instead of characters, we work with covert features of some sort, but still including one that determines when we move to the next word, we presumably get a similar distribution. But the models we actually work with rarely, if ever, predict that language will have this striking and invariable statistical property.

Time to return to serious things. For me, the event that most clearly marked the birth of computational linguistics was the invention by John Cocke in 1960 of what I always took to be the first context-free parsing algorithm that allowed for ambiguity. If it indeed was the first—and that turns out not to be beyond doubt—then this was the first algorithm designed expressly to meet needs that arose in our field. This happened at Rand just before I got there, and it became a source of great excitement for me.

The algorithm works only with binary rules, that is, rules with two symbols on the right-hand side. In its simplest form, the algorithm is based on a triangular matrix, which we can call a **chart,** with a box for each substring of the string being parsed. The idea is to fill the boxes one by one in such an order that the boxes containing potential constituents of the phrases in the current box will already have been filled. We assume that the boxes corresponding to single words are filled in an initial dictionary look-up phase. The remaining boxes are filled in order of increasing length of the corresponding substring, thus maintaining the required invariant. Several other regimes would have the same effect.

The original algorithm was embodied in a Fortran program with five loops, as follows:

1. for *Length* from 2 to *string.length*
2. for *Position* from 0 to *string.length* − *Length* + 1

3. for *FirstLength* from 1 to *Length* − 1
4. for *FirstConstituent* in *Chart*[*Position*, *FirstLength* − 1]
5. for *SecondConstituent* in *Chart*[*Position* + *FirstLength*, *length* − *FirstLength* − 1]

The maximum number of iterations of the first three loops is determined by the length of the sentence, and this corresponds nicely with the observation, made later, that the time complexity of chart parsing with context-free grammar is $\mathbf{O}(n^3)$ where n is the string length. The number of iterations of the last two loops, on the other hand, is controlled by the number of items that there could be in a single box in the chart which, in the original algorithm, could grow exponentially with string length.

Ron Kaplan and I recognized that, with only very minor adjustments, this algorithm can be turned into one with the $\mathbf{O}(n^3)$ time complexity I just mentioned. One way to do this would be to give the chart an additional dimension so that, for each substring of the input, there comes to be a separate box for each grammatical symbol. The boxes contain the various structures that the given substring has, and whose top node is labeled with the corresponding symbol. The parser never needs to rehearse these different structures because, for the purposes of building larger structures, they are all equivalent. This means that loops 4 and 5 are no longer controlled by the length of the string, but only by the number of nonterminal symbols in the grammar. What could be more inspiring than this elegant algorithm to a young person trying to see a little bit of rationality in an otherwise apparently random field?

Another change that we made to the original algorithm involved the introduction of partial phrases, sometimes known as *active edges* because they could be thought of as on the lookout for complete edges that they could absorb, thus creating either a new complete edge or a partial edge that was nearer to completion. Since one can obviously construct a phrase of arbitrary size by assimilating words one by one to partial phrases, any algorithm that works only with binary context-free rules can easily be turned into one that operates with arbitrary rules. More importantly, partial phrases enabled us to break loose from the rigid regime of loops embedded in a particular way, effectively replacing the algorithm with what I came to refer to as an *algorithm schema* (Kay, 1982). The basic idea was this: At any given moment, a phrase, or partial phrase that had been recognized, was stored in just one of two data structures, which we referred to as the *agenda* and the *chart*. If it was on the agenda, then the possibility that it might be extended by absorbing other edges had not been explored. But if it was in the chart, all such interactions with other phrases already in the chart had been systematically explored. The parsing algorithm consisted in moving phrases and partial phrases from the agenda to the chart in such a way as to maintain this invariant. In other words, repeat the following cycle until the agenda is empty:

1. Remove an arbitrary partial or complete phrase from the agenda.
2. Locate all current items in the chart that it could either absorb or that could be absorbed by it, putting all resulting new phrases and partial phrases on the agenda.

In 1974, Ron Kaplan and I joined Xerox PARC, and before long, I had an Alto computer in my office with 64K of memory just for me. This was one of the machines that Steve Jobs saw during a legendary visit. My intellectual history, if I may use such a pompous term, is studded with programming languages, and one that I encountered

soon after arriving at PARC was Prolog. Yes, I encountered Smalltalk, but Prolog made a more lasting impression, and I still use it today. Prolog is a way of extracting generalizations from algorithmic problems which, if they are just the right problems, cannot be done as effectively in any other way. Here, for example, is a top-down parser.

```
parse([], String, String).
parse([Goal | Goals], [Goal | String0], String) :-
  parse(Goals, String0, String).
parse([Goal | Goals], String0, String) :-
  rule(Goal, Rhs),
  parse(Rhs, String0, String1),
  parse(Goals, String1, String).
```

It defines a single predicate, called `parse`, which is true of a sequence of *goals* and a pair of lists of words or phrases if the first list of words and phrases has a prefix that can be broken into sublists that match the goals in the given order, and the remaining suffix is identical with the second string. Notice that, if there is a single goal, namely, *Sentence*, and if the third argument is the empty list, then the second argument must be a sentence. Given particular arguments, there are three ways in which we can attempt to show the predicate to be true of them; hence the three *clauses* that make up the program. Here is what the three cases do:

1. If the list of goals is empty, then they can be met only by the empty string and the second and the third arguments are identical.
2. If the first goal is met trivially, by matching the first item in the list that is the second argument, then if the remainder of that argument is identical to the third argument, the clause succeeds.
3. If the grammar contains a rule that would replace the first goal by a sequence of other goals, and if parsing some prefix of the second argument meets these goals, and if, furthermore, it can be shown that the remaining part of the string meets the remainder of the initial goals, then the clause will have succeeded.

This is a Prolog implementation of the classical recursive-descent parser, or top-down left-to-right parser, celebrated for its inability to handle grammars with left-recursive rules. The Prolog implementation reveals in a unique way the similarity between this parser and the following bottom-up left-corner parser:

```
parse([], String, String).
parse([Goal | Goals], [Goal | String], String) :-
  parse(Goals, String0, String).
parse([Goal | Goals], [First | String0], String) :-
  rule(Lhs, [First | Rhs]),
  parse(Rhs, String0, String1),
  parse([Goal | Goals], [Lhs | String1], String).
```

There are only minor changes, and they are all in the third clause. Instead of looking for a grammar rule that expands the next goal, we look for one with a right-hand side whose first item matches the first item in the second argument. If such a rule can be found, we treat the remaining items as goals that we try to find in what remains of the second argument. If this can be done, we can replace the matching sequence with the single symbol that constitutes the left-hand side of the rule, and we attempt to meet the original set of goals with a string modified in this way.[1]

The parallelism exemplified is surely elegant almost to the point of being beautiful and, in the design of algorithms, as Richard O'Keefe said, elegance is not optional (O'Keefe 1999).

Ron Kaplan had spent a lot of time working on augmented transitions networks (ATNs) before coming to PARC and, when we got together again, I became interested in them also. They seemed to me to have the power one needed for syntactic analysis, but they lacked a property that I thought crucial, namely reversibility. An ATN that did a good job in analysis was of no use when it came to generation. An ATN, as you doubtless know, is different from a standard finite-state automaton in two key ways. First, the condition for making a transition in a given network can be that the symbols starting at the current state in the string are acceptable to some other network named in the transition. In other words, they are recursive. The second difference is that networks produce output by making assignments to variables, or *registers*. The collection of these registers at the end of the network traversal constitutes the output.

Consider a greatly simplified example. Suppose the *Sentence* network is applied to the string *The book was accepted by the publisher.* We can assume that one of the transitions from the initial state calls the *Noun Phrase* network, which recognizes the first two words. The transition in the *Sentence* network puts the phrase in the *subject* register and moves to the next state, where one of the transitions allows a part of the verb *to be*, which is put in the *verb* register. At the next state, one of the possibilities is the past participle of a transitive verb. The system now abandons the search for an active sentence and proceeds on the assumption that it is passive. However, the work that has been done is not abandoned. The noun phrase in the *subject* register is simply moved to the *object* register because, if the sentence is passive, it will presumably fill the role of deep object. The subject register is cleared at this point. The *Sentence* automaton will presumably now be in a final state because the sentence could end here. But, if the word *by* follows, and then another noun phrase, this latter goes into the *subject* register.

Notice that, given the contents of the registers as they would be at the end of this process, it would not be possible to follow the same path through the network and generate the original string. To start with, *the publisher* would be in the subject register, and not *the book*.

In retrospect, it seems that the answer to this problem should have been obvious. But it is often so. Ron noticed that the key thing here is that you must never replace substantive contents of a register with a new one. If the register is still empty, you can put something in it, but once it has a value, that value must stay there. In other words, registers must be variables, in the mathematical understanding of that term, and

1 No award acceptance speech would be complete without a homework problem. So the question is: Why is it important to write `[Goal | Goals]` in the third clause, and not simply `Goals`?

not Fortran variables that can be assigned and reassigned indefinitely. And so, he proposed what he called the *same* predicate. It meant:

1. If the contents are the same as the argument to the predicate, continue.
2. If the register is empty, and the current item in the string matches the predicate's argument, put that item in the register, and
3. otherwise fail.

On the first branch, it seems that we are indeed dealing with a predicate. On the second, we have an assignment operator, and on the last, a predicate. This mixture is now quite familiar and we refer to it as *unification*.

In everyday terms, unification can be thought of like this. The CIA sends out a couple of observers to write reports on people they are watching, and given their reports, the question arises as to whether they could be watching the same person. One report says that the person's eyes are blue, and the other one does too, so they could be watching the same person. One says the subject's hair is black or brown and the other one says it is brown or red so, if we assume that it is brown, they could still be watching the same person. Furthermore, we have more specific information on the person's hair color than either observer gave us individually. One report says the person had an Italian accent. The other one does not have any idea, so we will take it that he is Italian. As long as the properties remain compatible with one another, we take the most specific version of that property that you can and add it to the output. As soon as they become incompatible, we decide that the reports cannot be about the same person and the process fails.

So much for what unification is. Now what about the name? Here is the true and unblemished story. I formulated the scheme I have just outlined and wanted a name for it. Until it fails, the process is conflating information from the two observers and is thus somewhat like the union operation on sets. But it is not exactly union, because it can fail. So I settled on a name like "union", namely, *unification*. However it was soon brought to my attention that this word already had a meaning in logic programming. So I started working on two problems in parallel. I started looking for another name and at the same time, I started trying to find out how my first choice was used in logic programming. To my great surprise, I discovered that, although some of the details were different—in particular, the logic programmers did not have the attributes that I had taken over from ATNs—we were doing essentially the same thing. So I did not have to give up the word.

Out of this, there came a whole new view of how the deep structure of a sentence might be related to its surface structure, or how its predicate–argument structure might be related to its tree structure. The dominant view—and it is still dominant today—is that you produce a deep structure, of the same data type as the surface structure, and you carry out a number of transformations. They may not be specified by transformational rules, but they are changes, perhaps in accordance with some principles, that replace one structure by a different one of the same general kind. When the sequence of transformations comes to an end, what is left is the *surface structure*. This scheme, as you know, has been complicated in a variety of different ways that need not concern us. According to this view, the relationship between the deep structure and the surface structure is an essentially procedural one. Now, a computationalist never wants to be handed a set of procedures designed by someone else and asked to implement it, because the procedures may

be hard or impossible to implement, or to reverse, or even to understand in usual computational terms. The computationalist wants to be given a declarative statement about relationships and allowed to work on the procedures himself; that is his job.

What came out of these considerations is a view that there is one data structure that represents both deep and surface structures. It is a tree[2] whose nodes have complex labels, and this object as a whole more or less corresponds to the surface structure. The topmost label of the tree, however, has enough articulation to constitute the entire deep structure. So the label will not just simply say 'S'; it will say that the structure represents a sentence and, furthermore, that it has this subject and that object, that the subject is definite and singular, and so forth. So all that led to unification grammar.

Another thing we worked on at PARC was finite-state technology, something that is probably still hot enough for many of you to be quite familiar with it. We noticed that it is a fact about regular languages, which are generated and accepted by finite-state automata and which occupy a position almost at the bottom of the Chomsky hierarchy, that they are closed under the operations of set theory and concatenation. This means that, if you can write an algebraic expression that describes the language that you are interested in, then the computing of it will be straightforward and can be specified algebraically.

Regular relations, which are modeled by finite-state transducers and are very closely related to finite-state machines of the standard kind, have just the power that is needed for many linguistic operations, particularly at the low end of the hierarchy—phonology, morphology, and spelling rules. In particular, as Ron and I pointed out (Kay and Kaplan 1994), a slight adjustment to the rules of engagement, as they are called in military circles these days, moves simple string-rewriting rules right from the top of the Chomsky hierarchy to the bottom or, at least, one step from the bottom. The rules I have in mind are of the form:

$$\alpha \rightarrow \beta / \gamma \_ \delta$$

meaning "replace α by β if there is γ on the left and δ on the right". If we stipulate that no rule be allowed to rewrite any part of the string that is part of the output of a previous application of that same rule, and if the rules are ordered, then the set of them can be modeled by a finite-state transducer that we can construct automatically from the rules. In this form, they are highly efficient to apply and reversible.

My professional life almost encompasses the history of computational linguistics. But I was only fourteen when Warren Weaver wrote his celebrated memorandum drawing a parallel between machine translation and code breaking. He said that, when he saw a Russian article, he imagined it to be basically in English, but encrypted in some way. To translate it, what we would have to do is break the code, and the statistical techniques that he and others had developed during the second world war would be a major step in that direction. However, neither the computer power nor large bilingual corpora were at hand, and so the suggestions were not taken up vigorously at the time. But the wheel has turned, and now statistical approaches are pursued with great confidence and disdain for what went before. In a recent meeting,

2 Actually, it usually allows reentrancy and is therefore not strictly a tree, but this need not concern us here.

I heard a well-known researcher claim that the field had finally come to realize that quantity was more important than quality.

The young Turks blame their predecessors, the advocates of so-called *symbolic* systems, for many things. Here are just four of them. First, symbolic systems are not robust in the sense that there are many inputs for which they are not able to produce any output at all. Second, each new language is a new challenge and the work that is done on it can profit little, if at all, from what was done previously on other languages. Third, symbolic systems are driven by the highly idiosyncratic concerns of linguists rather than real needs of the technology. Fourth, linguists delight in uncovering ambiguities but do nothing to resolve them. This is actually a variant of the third point.

This is a bad rap, and the old-school computational linguists who have made such a resounding success of their field should not take it sitting down. Let me say a quick word about the first three points and then expand a little more on the last.

First, robustness is an engineering issue. To throw out the theory because of inadequate engineering is to throw out the baby with the bath water. There are many approaches that could, and should, be taken to this problem, some statistical, and some not. Second, one of the things that linguists know that often surprises others is that the similarities among languages are much more striking and important than their differences. They can and do profit from these insights. Third, what look like the cute examples and arbitrary infatuations of linguists often, though not always, represent a distillation of important and wide-ranging issues.

Now I come to the fourth point, which is ambiguity. This, I take it, is where statistics really come into their own. Symbolic language processing is highly nondeterministic and often delivers large numbers of alternative results because it has no means of resolving the ambiguities that characterize ordinary language. This is for the clear and obvious reason that the resolution of ambiguities is not a linguistic matter. After a responsible job has been done of linguistic analysis, what remain are questions about the world. They are questions of what would be a reasonable thing to say under the given circumstances, what it would be reasonable to believe, suspect, fear, or desire in the given situation. If these questions are in the purview of any academic discipline, it is presumably artificial intelligence. But artificial intelligence has a lot on its plate and to attempt to fill the void that it leaves open, in whatever way comes to hand, is entirely reasonable and proper. But it is important to understand what we are doing when we do this and to calibrate our expectations accordingly. What we are doing is to allow statistics over words that occur very close to one another in a string to stand in for the world construed widely, so as to include myths, and beliefs, and cultures, and truths and lies and so forth. As a stop-gap for the time being, this may be as good as we can do, but we should clearly have only the most limited expectations of it because, for the purpose it is intended to serve, it is clearly pathetically inadequate. The statistics are standing in for a vast number of things for which we have no computer model. They are therefore what I call an "ignorance model."

Finally, a very quick word about machine translation. The days of the *ALPAC Report* are long gone, and there can no longer be any doubt that there is a need for machine translation. There are two kinds of people who need machine translation. There are people who need it because they need to disseminate documents in more than one language. The European Union needs to produce material in 20 languages, either because it has an operational need for it in 20 languages, or because the law says it must be available in 20 languages—not always the same thing. So it must be translated, and the result must be readable. Some of it has to be *very* readable because, in most cases, a document that has legal force has the property

that, if you have to go to court, you can choose which one of those 20 versions you are going to base your case on. Caterpillar Corporation produces huge amounts of documentation—almost more weight of documentation than of bulldozers—in an average of 14 languages. These have to be high-quality translations. What is required in these situations is entirely different from what people need who are consumers of translation. The canonical examples of these are people who are concerned with homeland security or people at Google. They are interested in anything you can tell them about a document. If you can't tell them anything, well that is too bad. If you can tell them a little, then they will be grateful for what you can tell them. Any kind of translation is better than no translation at all.

Not surprisingly, what the very word "translation" means for these two sets of people is entirely different. And I just would like to hope that you, the computational linguists of the future, will keep in mind the needs of both of these very worthy communities.

So, just a couple of final reflections. Statistical NLP has opened the road to applications, funding, and respectability for our field. I wish it well. I think it is a great enterprise, despite what I may have seemed to say to the contrary.

Language, however, remains a valid and respectable object of study, and I earnestly hope that the ACL will continue to pursue it.

We have made little headway in computational psycholinguistics, which to me has always been the nub, the center, the thing that computational linguistics stood the greatest chance of providing to humanity. To build models of language that reflect in some interesting way on the ways in which people use language. There has been some wonderfully interesting work on such matters, but not nearly enough. I am sorry that it has not been pursued as earnestly as I think it could have been, but it is a difficult field and perhaps that is enough reason in itself. My friends, I have spent some 40 years with you in this association, and I hope to spend many more.

References

Kay, Martin. 1982. Algorithm schemata and data structures in syntactic processing. In Sture Allen, editor, *Text Processing: Proceedings of Nobel Symposium 51*. Almqvist and Wiksell International, Stockholm. Reprinted in B. J. Grosz, K. Spark Jones and B. L. Webber, editors, *Readings in Natural Language Processing*, Morgan Kaufmann, 1986. San Francisco.

Kay, Martin and Ronald M. Kaplan. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3), pages 331–378.

O'Keefe, Richard. 1999. *The Craft of Prolog*. MIT Press Cambridge, MA.