NaturalLanguageProcessing-Lecture15

Instructor (Christopher Manning):Okay. Hello. And so today for 224n, I'm gonna start into the topic of looking at semantics. That is the assured meaning on how to understand text. I think it's fair to say that, in general, when people think of natural language processing and what it can do for them really what they want is okay, text is gonna come in and we're gonna understand it, the meanings are gonna come out. It's good that I'm eventually getting to the part of the course to talk about that and say something about this. There's some kind of technical class organizational reasons for why I don't say so much about natural language understanding in this class, which is partly that every second year Dan Jurafsky and me have been co-teaching a separate class on natural language understanding.

It's also partly a reflection of the fact that if you actually look at what's been happening in the world over the last decade that a lot of the work that's actually being used and being successful hasn't really been trying to do high level natural language understanding. A lot of the time is actually being focused much more down, doing the kind of lower level analysis tasks that we've spent a lot of the time on. It's also clearly the case that those tasks are, sort of, useful perquisites for being able to do deep and natural language understanding.

Nevertheless, thinking globally, I mean, what everyone wants is to watch their science fiction movies and to be able to have their pet robot that will talk to them and that seems to involve being able to do natural language understanding. A lot of the other applications you can think of for natural language are doing summarization, translation, question answering, spoken user interfaces. All of these you actually have to understand what's being said and what's being talked about. We have to act intelligently. It still seems to be the case that if you're thinking ahead to what can be done in the next decade as the new kind of intelligent applications that can be delivered. It seems like it's crucial to be able to deliver more intelligent applications than we have at the moment and that they actually have to understand more of the material that's available. So something like web search, it works pretty well a lot of the time, but if you think about how could I offer a qualitively better experience of what's going on it seems like at some point you actually have to have a system that can understand more of what's on those web pages, so it can do more intelligent things and be able to take actions. On the other hand, I mean, that's kind of being a viewpoint that's time has, sort of, never quite come.

If you look back on the history of AI it seemed completely obvious to people in the 1970's that if you're gonna do anything useful with natural language understanding and processing then you had to have understanding in deep semantic representations and be able to process those to do natural language understanding. I really think that the most dramatic thing, in summary, of the last decade is effectively that people have shown that you can just get amazingly far in producing useful artifacts by understanding almost nothing. That that's really been the breakthrough in the deployment of natural language technologies in recent years.

But, nevertheless, I don't think that that buys you everything. A kind of low level processing techniques are appropriate when you want to deal with speed and volume and you can get away with doing a very, very limited amount of semantic understanding to be able to address some tasks. You know, there are lots of good tasks out there. I mean, lots of things that people would like to have done for them of having their spelling corrected, finding things on the web, extracting who works at what company, processing people's resumes. There are tons of good tasks out there that people are now working on and can do a very good job with very little understanding.

There are also just lots of tasks in which you just have to understand much more to be able to attack them sensibly. I think, actually, one of the key distinctions will effectively be when you want to be able to have computers form tasks where they have independent responsibility for doing low level actions, which might build up to a higher level action. So for something like web search the computer finds candidate documents, but the human is right there to look at the documents and decide which documents are good ones and which ones are bad ones and they read the documents and get whatever information there is to be gotten out of them.

Whereas if you have your computer that's responsible for higher level tasks like the computer is responsible for going out and investigating what kind of an espresso machine that you should buy and then placing the order for the best price. Well, then the computer really has to understand a lot more about what's being said on pages to understand what's a good espresso machine to buy. Okay.

People are starting to do deeper forms of natural language processing and natural language understanding in the context of the kind of information access that's been so important in recent years. One obvious venue for that, and one that I'll spend some time talking more about in the later class, is the idea of question answering. The idea of question answering originally came out of the information retrieval community as what's a way in which we could give a richer form of information access. The standard information retrieval is an incredibly dumb form of information access. That people have something that they're interested in, information on, and the system says I can shovel you some pages that I think are relevant to that information need, but clearly there are much more profound and focused ways of delivering information to a person in response to an information need and one way it seems to cover a bunch of cases is people have some fairly specific question in mind and you can actually give them the answer rather than just shoveling them pages. People have worked a lot on that in recent years and in various times people have surfaced some of that in search engines. So here's a screen shot from about three years ago, I think, of MSN Search where I can ask it the question of which is the largest African country and it actually just gives me an answer. Answer Sudan, republic in northeastern Africa, the largest country of the African continent, blah, blah, blah, blah, blah. Then one of the questions is how big a need does that fill. I mean, when MSN Search was first announced that this was one of the things that they featured as one of the things they distinctively did. If I skipped forward to a more recent screen shot of Live Search and I give exactly the same query of which is the largest African country I'm now just getting web search results. It's not the case that they've completely turned off

that functionality because if I ask what is the capital of Sudan? I am still getting this Sudan capital, Khartoum, relation extraction kind of answers coming back. Though you might wonder about immediately after that then is this a useful link? Which is, I guess, I'm doing a little bit of research on the side to decide whether this is actually the kind of functionality that people want to have or not.

At any rate, I mean, it's very easy to then move into spaces where you just ask slightly richer questions and these systems aren't giving you answers that are kind of real answers. Which countries does the Danube flow through? That's then not being recognized as any kind of special question and it's just giving me web search results. One of the interesting things, and this is why web search engines have just been so, so successful, is it turns out that providing you're in a situation where the human can just look at things and interpret the answers rather than the machine having to automatically take the next step itself.

A surprising amount of the time that web search just works, right? Because if we look at these results, I mean, here we go. The Danube flows through nine countries, Germany, Austria, Slovakia, Hungary, and while the snippet could be a bit more perfect, but it seems like if I click through right there from the first result I can get the list of answers. This is just an aside, but if you look down a little at the results you see that the third answer says the Danube flows through ten countries. So there's some interesting issues on information provenance there as to who you want to believe when you're going on the web, but, at any rate, you're clearly very directly finding answers inside the web search engine. Okay.

Since I wanted to be keeping up with the latest that's going on, I also tried that this morning in power set and what is the largest African country? And it turns out that I'm actually getting just no particular value out of that. I mean, it's just giving me web search results as well and it seems like the web search results it's giving are actually worse than the ones that I was getting from Live Search in telling me the answer to the question in its snippets. Okay. Okay. So the comic version of what happens with our current natural language understanding systems corresponds to this comic of what we say to dogs. "Okay, Ginger, I've had it. You stay out of the garbage. Understand, Ginger? Stay out of the garbage or else." And what our computer system understand is "blah, blah, Ginger, blah, blah, blah, blah, Ginger, blah, blah, blah, blah, blah."

But the funny thing is that although our computer systems understand kind of in a sense nothing that there's this funny sense in which a lot of the time with a lot of the kind of bag of words representations that we deal with they actually do a surprisingly good job at answering these things. What I thought I'd just, sort of, do for a few minutes then is go back in time and tell you how the world used to be. For it's, sort of, a funny way in which in recent times there's been all this exciting work in doing machine learning and building systems up from large copper and lots of probabilities and abilities to do interesting things at a large scale and that work very well over large amounts of data.

At the same time does that work? I mean, that's kind of led to a sort of everybody forgetting how the world used to be and how in the old days people did used to build systems which were much more hand built systems that actually could in various ways do interesting semantically precise processing. So Fernando Pereira is these days a well-known machine learning probability in OP researcher. He's one of the people centrally involved in conditional random fields and I briefly mentioned doing MEMM's and he's actually working at Google right at the moment.

But way back when, in 1980 Fernando Pereira was working on his Ph.D. dissertation and what he worked on then was this prolog system, which he called Chat-80 because it was written in 1980. Its goal was to do full natural language understanding in a restricted domain, so it can do question answering. There have been several kind of famous systems of that era that have been kind of key, kind of natural language understanding systems that were developed, but it was one of the very well-known ones. Which actually had a surprising longevity of usage. I mean, the most recent use I know of is I was sitting in a conference in 2000, 20 years after this was written, and these people were presenting a paper about a system where you could ask questions about train routes and schedules and it turned out that what they were actually using on the back end to do their processing was still a version of Chat-80 and that they had sort of done development of a different Lexicon and grammar for that system. Okay.

So I'll just show you Chat-80 for a few minutes. This is something where just real soon now I'm gonna have to do some remedial work because it turns out the version of prolog I have for this only runs on Spark machines and we've just about reached the point where Stanford has gotten rid of their very last Spark machine. I guess for next year I'll have to get hold of a version that works on Linux, but fortunately for me I've still got until June 16th to be able to give this demo on a Spark machine so I'll do that. No, wait, I have to load it. That's a good idea. Okay.

So this has a database of geography. The idea is that you can ask various kinds of questions about geography and it will give the answers. So this is kind of America. Which countries border Iraq? Told about how bad geography knowledge is in the United States. Which countries border Iraq?

Student: Syria and Kuwait.

Instructor (Christopher Manning):Syria, Kuwait.

Student: Iran.

Instructor (Christopher Manning):Iran.

Student: Turkey.

Instructor (Christopher Manning):Turkey. Was that a vote for Jordan? Any more? We're up to five. Okay. Iran, Jordan, Kuwait, Saudi Arabia, you missed Saudi Arabia.

Okay. And then you can ask more difficult questions. You can say which countries that border Turkey border Iraq. Okay. What's the answer to this one?

Student: Syria.

Instructor (Christopher Manning):Syria, yeah. Any others?

Student: Iran.

Instructor (Christopher Manning):Iran was that? Yeah, Iran and Syria. Okay. There are lots of other kinds of questions that you can ask it. I mean, you can ask it what is the total population of countries south of the equator. 409 million. It's probably grown a bit since 1980. So one of the things that you'll note for this is it's not very up-to-date on modern world stuff. This is still in the era when the Soviet Union existed. All right. So I can ask which countries border the Soviet. And it also has a little cheat that for multi word things it doesn't identify as multi word things and you have to use an underscore. You can ask all the obvious stuff. What is the capital of England? It knows about that. You can kind of make them more complex.

So I could take my earlier example. Total population of countries south of the equator and not in South America? Right. No, it didn't like that one. What is largest country in South America? Brazil. Yeah. I can ask that. I'm not quite sure why the one I just asked got and I don't understand, but that's absolutely a property of the system that I will not deny for a moment. This system has got a fix grammar and a fix Lexicon and providing you stay within the boundaries of that you can ask quite complex things and it really does manage to work them out okay, but if you do anything that goes outside the bounds of that grammar by some means then it's just no dice and you get I don't understand and that's the end of the story.

So you can ask it to give more information about how it does all of this. You can ask for a trace, but maybe before I do that I'll just show you a teeny bit of what's there. So effectively, what does this system run on? That there's little databases where the little databases are data log for those of you who have done your 145 stall class. So here are facts about countries. So you have these predicates about countries where you have the country, the regency in latitude, longitude. That's why you can ask about things like south of the Equator. Area, population, capital, and currency. These are facts about countries and then there are some other databases that tell you facts about other things and there's a top level one of world zero dot PL where various relations define and various other things.

In particular, what you can see here is some of the kind of access of predicate. These access of predicates are kind of like having views on the database. If you want to know whether something is a country you're basically seeing whether it fills the first slot of the country with relation or not. You can use adjectives like African and that's then being realized as an incontinent relation and you can work that out from those relations.

Relationships are worked out like that. Top level is then Lexicons and grammars. Get the files going. Okay.

So that the actual – okay. Here's essentially where it's working out a Lexicon of different words and this then. A sentence can be a declarative sentence with a period, a WH question, imperatives. You then have how to expand out a WH question. Then down here we had structures and down phrases, so it's got a very explicit grammar like that. Okay. So once I have trace on it if I then ask it a question like which countries border Mexico? I mean, what we get is that what the system does is it builds up a paths tree, so this is showing a paths tree. It's an indented list representation. It's a WH question, which has the WH variable out the front here. The thing that we're gonna want to find is the which and then there's a sentence with a noun phrase, so it's plural.

Which countries and here's the country. And then it's the verb border and then you have the argument MP that's the object Mexico. It builds up a form of syntactic paths tree like that. Then what it does is converts that into a semantic representation answer. It's a country and it borders Mexico. It actually has a little query planner in there like a database system, which decides what's an efficient way to answer this query. It works out how to plan the best query result and then eventually here we get our answer back again. Okay. Maybe I should just show one other type of question. You can actually ask list questions in it as well.

You can ask something like what are the capitals of the countries bordering the Baltic. It'll do all that stuff and then we can actually get back a list here. So Denmark's capital Copenhagen, East Germany's capital East Berlin, etcetera. I guess they're also yes/no questions. Does Afghanistan border china? Is it a yes or a no?

Student: No.

Instructor (Christopher Manning):No, that's wrong. It's a yes. Yeah. There's a little bit of China that sticks off. Okay. So that's the Chat-80 system. I'll do one more. Which is a long one that I knows work providing I can spell it correctly. Which country bordering the Mediterranean borders the country that is bordered by a country whose population exceeds the population of India? Wait, I didn't get it right. Eventually I can answer that question. Maybe I'll just leave it now and go on. Okay.

Provided you stay inside the grammar of quite complex questions and that kind of structure it can give answers to it. Okay. This is just, again, showing the kind of predicates there and then how you can have one of these effective views on the database where you can get facile about things. Okay. That's then kind of the picture of how you can build semantically precise systems, which actually take sentence structures and understands stuff out of them. Some we'll spend today and then in more detail next Wednesday after the day off on Monday. Talking about how people build these kind of deep understanding full semantic systems.

In a way, this is a step away from as to what we do for the rest of the course since really the kind of methods that are used here are mainly logical symbolic methods of being used to develop these systems. There's just recently started to be some interesting work where people have started trying to combine some of the ideas of these kind of logical meaning representations with using machine learning methods to acquire them and infer them, but to some extent that that's still very recent. I mean, if you're actually at the level of saying, "Okay, I want to build a system that can actually do full sentence understanding in the way that Chat-80 delivered." It's not really that there's a lot of off the shelf technology to do that that's come out of the last 20 years of what people have been doing a lot of machine learning.

There are obviously ways in which technology has developed since Chat-80. People might give you a Gulliant face and people might have the system hooked up to a website that has facts on it rather than using a little prolog database, but there's kind of, in a sense, in terms of actually building a system that does full understanding of sentences like that. There's not a huge amount that's very fundamentally changed since Chat-80 was built. Okay.

The kind of space we're in is really similar to the space of when people were building compilers and doing interpretation of formal languages, except we're gonna take the same ideas and then apply them for natural languages. If you're building a compiler and you want to understand arithmetic expressions, well, what do you do? You path up your arithmetic expressions by having a syntax of those arithmetic expressions and then what you want to do is to be able to build up semantic interpretations for these arithmetic expressions. How do you do that?

Well, the conventional way that people do that is they have a grammar that will build up these kind of syntax trees and that's what you write in a tall like bison or yak or something like that. Then hanging off the walls of those grammars, you then have semantic attachments that say how to interpret the various pieces. You would be looking at this kind of syntax tree and here what we have is just constants 3+5*6. You have these semantic attachments, which would then say how to interpret these things. This means multiplication, this means five and six, and, therefore, here we can then evaluate this modification of five and six. The meaning that we evaluate to here is 30 and then we'd add three and the meaning that we evaluate to is 33. Hopefully that seems familiar.

That's the kind of easiest case where we have constants and so we can evaluate a compile time and just work out the meaning of that is 33 and put 33 into our system. I mean, the more typical cases that we write in arithmetic expression is something like this 3+5*x. Then what we want to do is say we're still gonna have semantic attachments off our syntax rules and those semantic attachments are gonna tell us how to work out the value of the whole thing. Now the meaning of the node can't be a fully evaluated answer and instead has to be a little piece of code that will tell us how to calculate the answer. Now, what we're gonna have is the meaning given by semantic attachment here. You're going to multiply by five whatever the value of this is that's determined when the program is run and then you're going to add three to that.

We can still work out a meaning that's a semantic attachment going up, but it's unspecified because we don't know the values of things. Okay. The general idea of what you're doing here in natural language processing, this is referred to as rule-to-rule translation. The idea is we have syntax rules, so here's our syntax rule. Expression can consist of an expression, a function, another expression, and then we have a semantic attachment attached to this syntax rule saying suppose you know the meaning of these three things. I'll tell you how to calculate the meaning of this bigger thing. All right. That's the semantic attachment and that's referred to as rule-to-rule translation because it's knowing these rules meanings. We then give a rule to method of calculating this rules meaning and you work compositionally up the tree. Okay.

Essentially the picture that we're gonna build is let's work out how to do that same thing for natural language sentences. In some sense, we'll refer to that as understanding the sentence. I mean, this concept of what counts as understanding is actually a very tricky philosophical issue. I'm not gonna spend more than one slide on it, but I'll just mention it. This means that we understand this arithmetic expression and we've calculated doing the semantics for it as to what these arithmetic expressions mean. There's a, sort of, sense in which that seems to be true, but in another sense you could say, well, no, you've taken one bunch of symbols and you've converted them into another bunch of symbols. Why is that understanding anything?

That's this question of then what counts as understanding. So, I mean, there are various ways that you can think of what understanding means. One way of thinking of semantics is operational semantics. If you can carry out actions that means that you understand things. If the person can type throw axe at dwarf and that happens, well, that means that you've understood it. In logical philosophical work a very common idea is this notion of true conditional semantics. If you can determine the truth of the statement relative to a world that means that you've understood it. Okay.

Similarly, if you can answer questions that means that you've understood it. For many years, people have suggested in natural language processing that ability to translate is proof of understanding. If you can take something and translate it into Chinese that means that you've understood it. If you start thinking about some of these notions they're very, very slippery, right? Because while we worked on machine translation and, okay, the machine translation systems that you guys built were very good, but hopefully your model two output of the decoder was getting to be almost intelligible sometimes. The bigger peoples statistical empty systems and sometimes they can translate sentences perfectly into different languages, but if you're thinking about what you built for programming assignment two you probably don't feel in your heart of hearts that your system understood the language.

It was just doing these probability tables and sometimes something okay came out at the end of it. It's very unclear what then counts as translation, but, nevertheless, the goal that we're gonna work with at the moment is this traditional goal of understanding means translating into formal logic. Essentially a first order prepositional calculus version, predicate calculus version. The picture that I'm then gonna show you next Wednesday is

that, at least in certain circumstances, will let us do this top high level thing of being able to perform tasks. We can then fairly readily convert those representations into something where we can quarry a database. We'll put stuff into a database, so that we have an operational task semantics. Okay.

Yeah. Today I'm gonna say some of the general ideas of compositional semantics as an introduction and then I'm gonna briefly mention at the end semantic grammars that are an alternative that have sometimes been used. Next Wednesday I'm gonna talk in detail about how you can build compositional semantic representations. Then for the final week of the – and then I've sort of been saying nothing about representation reasoning. Then for the final week of the class on Monday I'm gonna talk about lexical semantics, which is how do you actually work out what words mean because as well as you have to combine meanings you have to know what words mean. Then on the final class, I'll talk a bit about some of the high-level semantic applications that people have been working on of doing question answering, semantic search, textual entailment, and problems like that. Okay.

I know there's this problem that people who do computer science and similar subjects often end up feeling like they sit through five different introductions to logic, so I'm not really gonna do much of introduction to logic and assume that you've seen logic before. Basically what I want to work towards is how can you construct logical representations? Which, in general, in logic courses no one ever says anything about. Often in logic classes in the first two weeks there are exercises where you're meant to take sentences in English and convert them into logic. The way you do that is you use your brain in serious ways and write down what's the correct logical interpretation. Then for the rest of the semester you work with predicate calculus P implies Q of X or YZ or something and no one talks about human languages ever again.

Which is, in some sense, wrong. Historically why formal logic was invented was it was meant to help people understand human reasoning and human languages. That was actually the motivation, but it's sort of tended to be lost in the logical tradition when people have worked with formal representations. In particular, most of the time people just don't actually work out how can you do any kind of automatic translation from human language sentences to logical sentences. That's what we're gonna look at. Okay. Nevertheless, what are we gonna have underlined here? What we're gonna have underlined here is we're gonna Booleans, you know about those, we have individuals, so there are things like people, might be other things like times, and then we're gonna have functions of various times our predicates, so frog(x), green(x), and, in particular, we're gonna cause predicates once, which have a Boolean return value.

So is it green is the predicate. Okay. Predicates define sets who are the individuals that satisfy the predicate. If the predicate has only one argument, as a lot of the ones we look at are, I'm gonna call them properties. Essentially we can make more complex functions out of simple functions. In particular, with the kind of techniques that people use to represent natural language meanings and the construction of natural language meanings

tend to make quite a lot of use of higher order functions. Higher order functions are then functions that take functions as arguments and maybe return functions as their values.

To be able to do that we use lambda calculus. How many of you guys have seen lambda calculus in another class like programming languages? Some, but not all. Okay. When we write functions or methods in Java, right? That we give them names, so we have a name and it takes some arguments and then it maybe returns the value. Okay? We have a function name, right? But in some sense the key thing is the behavior of the function and we should be able to have functions that can do some things without giving them names. The idea of lambda calculus is it's a notation for being able to produce and process functions that don't happen to have names. The advantage of having functions that don't have to have names is that you can just invent them on the fly as you need them. All right.

You could write down a function that's a squaring function and you say it's sort of square of X and it's value is that it returns X*X, but you should also be able to just write a function that doesn't have a name. Here's the lambda function for swearing function. So you're writing a lambda before each argument of the function, then you commonly write a dot or a square bracket or something like that, and then you're saying the value of the function after that, which is what'll be returned. This takes one argument and calculates P*P and that's the value that gets returned.

What we're gonna be often doing for our natural language understanding is constructing these kind of lambda functions. Once we have a function like lambda P, P*P we can then take this function and do function applications, so we can give it an argument three. The way that we then evaluate this function is we substitute three for the argument P and then we evaluate this 3*3 and we return that as the return value of the function. Okay. That make sense? I can just hit spacebar and go on? Okay.

The crucial thing to note is in reality lambda functions only take one argument at a time. There isn't any inherent notion of functions that take multiple arguments in lambda calculus. What you do instead is you fake functions that take multiple arguments. The way you fake functions that take multiple arguments is you make use of this notion of higher-level functions. So suppose I want to have a function that multiplies two numbers. Well, the way I can have a function that multiplies two numbers is I can write a function that takes a number and what it's gonna do is return as its value a function. This function is gonna be a function that takes a number and then returns times XY, does the multiplication. If I take that function and then give it two arguments times(5,6) what's gonna happen? Well, what's gonna happen is I'm gonna take this function and apply it to the value five and it's gonna return as its value lambda Y*XY where X has become five. So that will be lambda Y*5Y and then I'm going to apply that function to the value six. So Y will become six and I'll return times(5,6). Okay.

I can make use of the idea of higher order functions to simulate multiple argument functions. Okay. That process is referred to as currying after a logician whose name was Curry, maybe not surprisingly. Okay. For our individuals we have some of them are

going to give names as constants, right? So that we can have a constant that's a person, which is George Bush. We can have a constant that is a named predicate, so whether something is red that can't be a constant. And then we can have constants that are kind of predicates and arguments. We have love predicate, where it's a known predicate of two arguments and we have facts in the world about whether people love each other.

Essentially we're going to assume that for things that we know about we have little database tables where we can look up who loves each other. Okay. Then we can go on from there and we can also then make representations of more interesting kinds of function. Here's the kind of interesting function that we can build. We might want to build a most function. Well, what we'd like to be able to say is most elephants are big. When we're saying most we have two predicates that are involved in the most representation. So we have the elephant predicate, which is what we're talking about, and we have the big predicate.

In both of those properties there are things that limit the set of individuals. They're the set of things that are elephants and the set of things that are big. We want to then take those two functions and then make a most function that operates over them. Here, again, we're into the space of higher order functions. Our representation of most is going to be that most is, I think it's kind of easiest to see here, but most is gonna be something that takes two functions, two properties, and then its going to itself be returning a Boolean value, which is saying is it true that most pigs are big? Okay.

So if we want to evaluate that how do we do it? Well, the way we do that is we're gonna have this. If we look at this function it picks out in our world a set of things that are pigs and this function picks out a set of things that are big and what we want to know is if you look at the set of pigs and then you ask which ones of those are big that over half of the things that are pigs – I had elephants, sorry, before. Whatever. Pigs, elephants. Over half of those things are big. Okay. Just like in our example of 3+5*6, we can write a little way to calculate whether things satisfy the most representation.

The crucial thing to note here, a slightly different way of doing things, is normally you don't see this when you do a predicate logic course. What you see are these two quantifiers, which I hope you remember well. Here's the four quantifier and there's the exist quantifier. That you are using this categorimic notation where you're introducing all this special stuff to represent the two quantifiers of their existence for all. When we're doing natural language translation by and large we don't want to do that. We want to be able to treat everything as a function. We want to have an all function and an exist function. The reason why we want to do that is because building functions is something we'll easily be able to do in our translation. Then stuff like this will be how we evaluate a function to see whether it's true.

The existing pig big function will be evaluated by looking at the set of big things and seeing if there's one thing or more in it that's a pig thing. It'll be a method of evaluating this function, so that's equivalent to this, but will just be seen as a method of evaluation. As well as the fact that we can generate these things to translate, the other reason we want

to do this is because natural languages don't only have their existence for all. There's, sort of, a sense in which those are fundamental for the purposes of logic, but natural languages have all kinds of quantifiers. They have ones like most and a few, but they also just have an infinite space of quantifiers. At least seven with a quite a large number for a small country.

You can make quantifiers of all kinds of sorts. Okay. The next thing you need to know about natural languages is as soon as you have quantifiers, quantifiers in natural languages have scope. In the semantic level it's kind of just like what we saw at syntax level with prepositional phrase attachments. We also have the semantic level you get these ambiguities of interpretation. Okay. Here's the funny Groucho Marx example. "In this country a woman gives birth every 15 minutes. Our job is to find that woman and stop her." This sentence has two possible readings. Okay. Which is the funny reading? The first one, right? There exists a woman and she gives birth every 15 minutes. And this is the kind of normal reading where for every 15-minute interval there's some woman who gives birth in it.

It's exactly the same situation of just like our PP attachments. The funny thing is, this is just the way natural languages are. That you get these scope ambiguities. Most of the time they don't cause people trouble, but if you're thinking for the formal understanding perspective basically whenever you have quantifiers you get a number of readings, which is exponential in the number of quantifiers and that's just how the world is. I'll show another example of that later. Okay.

But what I want to talk about for a moment is this method of constructing compositional semantic representations. Today I just want to sort of do a very simple example that gives the general idea and makes life look very easy, which makes the connection between the programming language semantic interpretation and what we want to do for natural language. Then next week we'll start looking at the cases where things get rather more complex and you actually have to be able to do tricky things with lambda calculus to make it work.

Our basic model of what we're gonna do is we get a sentence, we pass it through to get a syntax tree, we have a Lexicon where we can look up a meaning representation for each word, and then what we want to do is walk up our syntax tree and calculate a meaning for each bigger thing. The way we're gonna be able to do that is attach to each syntax role there's gonna be a semantic attachment saying how you can combine the meanings of the lower level parts to make the meaning of the parent load. If we start at the bottom and work up we'll be able to apply those rules and come up with a meaning for larger level units.

The idea that you can do that is referred to as the principle of compositionality. By in large, basically everyone believes in the principal of compositionality because if it weren't true natural language understanding would be impossible. We can construct new different sentences every day and people understand them. How can they understand them? They know what the words mean, they know what it means when you put them

together, and if you're saying more complex sentences like Sue claims that Bill said that Fred thinks that Joe acknowledges that Amanda claims that blah, blah, blah, we can just compositionally work it out in terms of the pieces.

On the other hand, compositionality isn't quite true in all places. There are various places from idioms onward in which there seem to be meanings for units, which aren't easily expressed in terms of meanings of the parts. Okay. So how can we realize this notion of having semantic attachments to syntax rules? One famous early way of doing this was the representation of definite clause grammars. Definite clause grammars are getting us back to prolog technology and what we saw for Chat-80. One of the cool things about prolog, if you're a natural language person, is that prolog came built into it as a syntactic sugar where you could write context free grammars and you just write down the context free grammars and then you could just pause lists of terms in terms of those context free grammars without actually having to do any extra work for yourself.

This is how you wrote down a context free grammar in prolog, which is fully equipped with what we've seen the context free grammars. Sentence goes to noun phrase, verb phrase. Noun phrase goes to proper noun. Noun phrase goes to its determiner noun. Very phrase, verb, noun phrase. Here's how you wrote the Lexicon. The items of the Lexicon were written as one element lists. If you just wrote that into prolog and then you asked the question sentence [list of John ate a lion] or something like that it would pause it and you get a paths tree. The thing that was slightly cooler than that about what you could do with these definite clause grammars that the definite clause grammars were actually being translated into terms, so that you had a term like sentence of X with arguments.

Rather than just having a plain CFG where your symbols were atoms, you could actually give your symbols arguments. Here's a slightly better CFG and this is a CFG that actually enforces subject verb agreement. I'm not sure if this worried anybody, but it's something that might have worried you when you were building your parses for the third assignment was the grammars that we were using didn't actually do anything to enforce agreement, right? If you gave them sentences like the man eat or lions eats they paths them perfectly well. They weren't attempting to check for agreement in any sense.

You might like a grammar that can check for agreement and see that you've got correct subject verb agreement. Here's how you can do this with a definite clause grammar. I can change my sentence rule to be adding now an argument, so I can say that the sentence consists of a noun phrase, which has some number and a verb phrase that has some number. Just by writing the same number there those two variables will be equated in prolog and both the noun phrase and the verb phrase will have to have the same number. Okay?

Then for the rule for the noun phrase and the verb phrase I essentially just inherit numbers. I'm saying the number of the noun phrase will be the number of the proper noun or the number of the noun and if there's a determinate it should share number as well. The verb phrases number will be the verbs number, doesn't matter what the noun phrases number is. Then down in my Lexicon I'm then saying facts about the number of

things. I'm saying that Mary is a singular proper noun, but can be either a singular or a plural determinate. Lion is a singular noun. Lions is a plural noun. Okay.

With just this grammar, I can now send it back into my prolog system and if I say the lion eats it will paths correctly because eats is singular, lion is singular, those will get inherited, and, therefore, this will match. Whereas if I try and give it the sentence the lions eats then this will be singular, that will be plural and so the top level S rule just won't match. It'll return that that sentence isn't accepted by the grammar. Okay. So that's kind of cool and so that leads us down to this space of writing feature based grammars because I can then start thinking of other things where I want to put features into my grammar and have those features checked as I paths.

One particular kind of feature that you can put into your grammar is you can actually then say, well, why don't I make use of the fact that I can put variables into my grammar and why don't I put variables in that actually have a meaning of the sentence. This is the idea of these semantic attachments. So here I now say I've got the sentences meaning and I'm gonna do rule-to-rule translation and how am I gonna do that? Well, I'm gonna have the noun phrase has some meaning and the verb phrase has some meaning and then the one other thing I could do with the DCG is put something after it in these curly braces and that meant this is the semantic attachment for this role.

The semantic attachment of this rule said combine the noun phrase meaning and the verb phrase meaning to give the sentences meaning. The way I do this combination is simply I do function application. I'm gonna take the meaning of the verb phrase as a function, apply the meaning of the noun phrase to it as an argument, and I'm going to return the result as the sentences meaning. Prolog was all written in relation late. There weren't actually any functions in prolog. Everything was a relation, but the applied predicate is saying take this function, apply this argument to it, and that sits return value and that's what I'm returning. Okay. So my grammar is just using function application everywhere to combine meanings here and then my Lexicon now has meanings for terms and my Lexicon is being written using lambda calculus.

John and Mary here are just constants, but for jumps I've got my lambda expression, so it's a function. Lambda X jumps X and for loves I've then got this higher order function that takes an argument and turns another function, which takes an argument, which returns loves as its value. If I've got this, now I can actually work out the meaning of the sentence. If you look at the meaning that's constructed from the sentence in my DCG and I want to have a meaning for John loves Mary, well, the Lexical items have some semantic translation. The name of the noun phrase just say our meaning is the same as our one child's meaning. This goes up and simile the verb. The first place something interesting happens is with the verb phrase.

The verb phrase of semantic attachment says take this function and apply this argument to it and my meaning is the result of that. You take Mary, Mary becomes Y, the return value is lambda X loves X Mary, and that's returned as the value. Then you do the same thing with John and you get loves John, Mary as then being returned as the meaning of

the sentence. That seems kind of hopeful. Now, we can work up the meaning of the two sentences John loves Mary – well, more sentences. John loves Mary, Mary loves John, and Mary jumps and John jumps. Okay.

We've got a four-sentence language and everything gets worked out correctly. This is essentially what Chat-80 was doing and the kind of technology we're gonna do. However, this case here is very easy. I've just taken names and represented them as constants and then they're just combined in the obvious way by a verb phrase as the function. All of the trickiness comes in because of the fact that a lot of the time natural languages just aren't that easy. Now, I want to say a word about that, but before I do there is this slide.

I showed a little bit of DCG notation there just because it connects up to Chat-80. To do this you don't have to use DCG notation. You can just say I've got a context free grammar rule just like before. I'm gonna put next to it a semantic attachment, which is saying how I'm gonna work up the meaning corresponding to the parent in terms of the meaning that it corresponds to the children. Just like my little baby example, normally the form of these semantic attachments for natural language grammars is inherently trivial. We identify one argument as the semantic head and it takes other children as arguments and that's essentially all they do. The semantic attachment is normally just function application.

All the trickiness actually comes down in the Lexicon of coming up with appropriate semantic representations for different words, so that you can combine the meanings together by function application. Okay. All right. Mary loves John is really easy, but the world gets more complex very rapidly. Here's just one more example of how the world gets more complex. A few years ago we actually worked a little on this system that still seems to me a cool idea. In the olden days, in the GRE exam there used to be an analytic section, which was multiple choice, which had various kinds of questions. One kind of which was logic puzzles.

They've now gotten rid of that and now have this incomprehensible analytic essay that I think no one knows what to do about in graduate school admissions. I think in the LSAT they still have these logic puzzles. At any rate, commonly computer scientists are the kind of people who when there were young did logic puzzles for fun at some point, so most of you have probably seeing these kind of logic puzzles. Here we have six sculptures. C, D, E, F, G, H are to be exhibited in rooms 1, 2, and 3 of an art gallery. Sculpture C and E may not be exhibited in the same room, dot dot dot. Then for each scenario there are sequence of questions, which in the GRE's start is the easiest one and got a bit harder.

I thought this was a cool problem. For the GRE, the hard part of this is meant to be solving the logic puzzle. That the natural language understanding is just a God given skill that all human beings have, but solving logic puzzles that is hard. Whereas if you think about this as a computer science problem, if you can represent the problem of the logic puzzle, solving it is trivial. All right? So all of these questions are over such small

domains that you don't have to do anything smart. You can just use brute force. You can just evaluate every possible assignment and see what's possible. If you paid attention in 221 you might think, well, this is like a constraint satisfaction problem. Maybe I can use ford changing or R consistency or one of those things and it's true you could.

You could come up with the answer in .001 seconds instead of .01 seconds, but really that part just isn't hard. The hard thing is how could you possibly take this natural language description and turn it into something that you can actually construct a model from, so that you could run your model checker or your R consistency algorithm to see whether it holds. This is, again, the kind of place in which we get into all of these troubles and translation to logic becomes difficult for all sorts of reasons actually. Just to mention one of them again that I all ready mentioned, this notion of quantifier scope and interpreting quantifier scopes.

Now, the people at the ETS who build GRE questions try really hard to make them unambiguous. That's part of their job because if the questions are perceived by regular people as being ambiguous then upset parents sue them and things like that, right? They really, really, really want to have unambiguous questions. The fact of the matter is that natural language you just can't do that because there are latent ambiguities. This is just illustrated for that example. The first sentence is at least one sculpture must exhibit in each room. Well, there are two quantifiers. At least one sculpture each room, so there has to be multiple readings, right? And there are, right?

There's one reading in which there exists at least one sculpture and for that sculpture necessarily in every single room that sculpture is exhibited. Now, that one's kind of ruled out because it's just impossible to satisfy in the real world and human beings don't pay that one that much attention. That means that somehow we have to be able to consider the semantic interpretations and work out which readings to exclude and settle on the right reading. If we then go on to the next sentence of no more than three sculptures may be exhibited in any room it turns out that we get more than two readings here because you also get extra scope ambiguities when you have elements like may.

May and various other modal operators, that they also introduce scopes. There are at least three readings that you can get for this sentence. One reading is for every room it's the case that there are no more than three sculptures that are exhibited in it. That's the reading that you're meant to get, but you can get other readings as well. These aren't readings that you can kind of rule out as impossible to satisfy in the world. Another reading is that, in total, at most three sculptures are exhibited. That's saying in any room whatsoever there can be no more than three sculptures exhibited and all the rest of them have to be kept in the attic or whatever. That's a possible state of the world. That's not impossible to satisfy. It's not what's intended.

A third possible reading for this sentence is to say that there's somehow a special set of three or less sculptures and that special set of sculptures can be exhibited in any room whatsoever, but for all the other sculptures there are restrictions on which rooms you're allowed to exhibit them in. Again, that's a possible state of the world, which can't be sort

of ruled out categorically. Somehow you have to be able to build and consider these representations and work out how to possibly evaluate them and so that starts to make the natural language understanding problem look much harder. In particular, none of those questions about scope were represented whatsoever in our syntactic representation of the sentence.

If we take the first sentence, at least one sculpture must be exhibited in each room, and we give it to our paths, the path is going to give a syntactic representation to that sentence. As we build our PCFG's, it'll just return the best one. Now, admittedly, there are other paths representations for that sentence, but the different paths representations aren't meant to be representing different semantic structures. All of these semantic differences are being represented at a level beyond the syntax. The syntax isn't really specifying all the details of the semantics. There are other things yet to figure out.

In general, it's the case that traditional syntactic grammars don't really represent the semantics of a sentence in a straightforward way. That they show the syntactic building blocks of that sentence, but it's just not their job to say, well, what is the scope of a quantifier? What's the scope of negation? Which things are predicated? Which things have the long distances? They're not doing that. Because of that, taking a syntactic grammar and then coming up with an interpretation in terms of semantics is a difficult thing. If you want to do that task, the way you do that difficult thing is you write complex lambda expressions and you write complex combination and imprint print rules, so you can get out semantic meanings even though that they can be quite distant from the syntactic form.

Next time I'm gonna talk about how to do that, but briefly just for the end of this time I want to mention an alternative. Some people think that that's much too much work to do. The alternative thing that you could do is just change your grammar, so that the grammar is trying to directly reflect the semantics. At that point you kind of give up on your grammar being a good representation of the syntax because you're effectively mucking with the grammar to make it look more like your semantic representations. This is known as semantic grammars.

Now, semantic grammar is a dumb name I've always thought because, I mean, a semantic grammar is just a grammar like any other grammar. You're writing a context free grammar. The semantic part is all in the person who wrote the grammars head. They're saying I'm gonna try and write the grammar to make it easy to get out the kind of meanings that I'm interested in. Where these grammars have been very widely used is where you want to do some limited kinds of semantic processing normally in restricted domains. They can be used in things like spoken dialogue systems.

Maybe I'll just go straight on and show the examples. Here's a famous old example of that from 1978 on the Lifer system, which was another system for natural language understanding. Its goal was to be able to answer questions about US Navy ships, so you can figure out who was funding this one. In the general space that this was in, and I guess also the space that Chat-80 was in, again, an interesting thing in the way the world has

gone. In those days doing natural language interfaces to databases was seen as a hot area and a place that natural language could be used. Clearly that's just not something that's taken off at all.

In practice, environments came along and everyone uses quarry builders and Microsoft Access or something like that and basically no one uses natural language findings to databases. Here's the idea of the semantic grammar. At the end of the day we have a task of we're going to be answering questions like what is the length of Kitty Hawk class ships? We could write a regular grammar that could path sentences of these kinds, and that's what Chat-80 did. Another way of doing things is to more directly look at the kinds of questions you want to answer and just write a grammar around those.

Here the grammar says a sentence is rewritten as present the something of ship, right? Present is you can say what is can you tell me or just tell me? I'm making no attempt to really understand, give a good syntactic structure to sentences. I'm just saying, well, there's various kind of ways you can ask questions about things. What is can you tell me, tell me. I'll just write them down. Attributes say things like link, beam, and clasp. Ship is the ship name. Here are some ship names. I have this kind of grammar that's specific to answering the restricted space of questions about this application.

Most of my categories aren't things like noun phrase anymore. Most of my categories are things like class name or attribute or this present thing. Words are recognized largely by their context. It's not really saying that they're nouns, but they're a kind of ship. You have this strongly directed recognition of semantic categories, but you don't have any kind of general reusable grammar. These systems are still commonly used. I mean, if you want to just do a restricted domain recognition task, writing these kind of semantic grammars for a restricted range of English is just the easiest way to do it. It comes at a high cost, since you have this very specific unreusable thing that's very directed to a particular application. Okay.

I'll stop there and next time come back to the general problem of how you can kind of translate syntax from the semantic forms.

[End of Audio]

Duration: 76 minutes