DAVIS: Throughout this series of lectures, one of the themes we've come back to repeatedly is the notion that in the knowledge lies the power. Well, that of course brings up an obvious question. If it's in the knowledge that the power lies, where then do we get the knowledge? That is, how are we going to be able to build up the stores of knowledge that we believe are needed in order to make these sorts of programs work?

More generally, the issue is of knowledge acquisition and knowledge base construction. How can we put together those knowledge bases? How can we collect the large stores of knowledge that we need?

Somewhat more generally yet, we're dealing with the notion of learning. That is, how can we have a program that learns? How can we have a program the builds up the appropriate store of knowledge that it's going to need in order to carry out the kind of problem-solving performance that we'd like to have? In today's lecture, we're going to look at one program called TEIRESIAS to try to explore some of these issues of knowledge base construction and maintenance.

What I've tried to do is see how it might be possible to build up a bridge between the expert who may know a great deal about a domain, and the program which is trying to learn more information about that domain.

And more generally, I said the issue was of learning. And as it turns out over the years there have been a number of different speculations about how that might happen. That is, gee, wouldn't it be great if only we could have a program that learned? If we could figure out how to make it learn, then we could just turn it on, come back again in a few years, and it would know all sorts of different things.

Well, there have been a number of dreams about how this might be accomplished. One of them is suggested here. Gee, if only we could take all the knowledge in the world, all the knowledge in all these textbooks, and somehow just kind of dump it into the program. Take all these books, let the program read through them, and in doing that, it would become very smart.

Well, clearly the problem here is dealing with the knowledge that's in those books. How can we get it into a form that the program could digest? That itself, of course, is a very difficult problem. And that's one of the things that keeps us from going off and saying, gee, all we have to do is take all of these books, somehow dump them into the computer, and magically it will learn all this information, synthesize it, put it together and get very smart.

So if that dream doesn't work-- if we can't give it books to read, can't make that practical-- is there another way we can do it? Is there some way we can just have the machine sit there and think great thoughts? Well, that's another dream that people have thought they might try. That looks something like this.

We might say that this is the 'contemplate your core' image of learning. Sit there and think great thoughts, trying to develop new concepts, and somehow learn how you might solve the problem by introspection, by speculating, by coming up with new ideas.

Well, there are the dreams. Now let's take a look at what the reality actually looks like. It's nice to think how we might make this happen, but somewhat more realistically, there are several different modes of learning that we can recognize.

First of all, we have the notion of learning by being programmed. Now it turns out, somewhat surprisingly I think, that almost all of us have been programmed at one point or another. And if you don't believe that, think back to the time of the years BC-- that is, before calculators-- and think back to how you learned how to extract a square root.

Now I'm sure many of you can remember the process you go through to extract the square root with arbitrary precision using pencil and paper. But I'm also sure that very few of you could actually explain that process, explain how it works and why it works. And in fact, you've been programmed. You've been programmed to carry out a certain sequence of operations that you don't really understand and yet you know will work, and can be made to work for you. So that's one mode of learning.

Another we might call learning by being told. That is, one thing I might do is I might listen to a lecture, look at a videotape, and learn about something from somebody who claims to know about that topic.

Yet another model of learning says that I might learn from examples. That is, if I can pick out an appropriate set of examples and hand them to a student, well then he might be able to learn in that mode. So in this fashion I might say, here's five examples of interesting expert systems. Here are descriptions of them. Go off and read about them. And on your own, figure out what it is that's interesting about them, what they have in common, what the important themes are there.

Notice how the student is doing most of the work here. All the teacher has done is pick out the appropriate selected examples and say, here are the examples you want to study-- these five things. But now it's up to you, the student, to pick out the important lessons there and figure out what the commonalities are.

Finally, and most ambitiously, we have what we might call the notion of learning by discovery. This is akin to that dream I was showing you a little bit earlier, but the point is, I'm not going to tell you what the appropriate examples are. Rather in this case, I'm going to leave it to you to figure out what the important concepts are.

As a result we simply try and set the program loose-- if there were a program capable of doing this-- have it explore some topic, have it look around, and it itself try and discover what the important notions were, what the important concepts were. We haven't selected out the examples. Rather we're letting the program try and traverse the terrain, if you will, the terrain of concepts, looking for the important ideas and the important concepts, zeroing in on those, trying to discover the important ideas.

Now, the program we're going to look at-- TEIRESIAS-- works primarily in the mode that I've suggested here. As it turns out, TEIRESIAS and MYCIN were invented somewhat contemporaneously, and in fact were designed to function together. MYCIN, remember, is the program that tries to do medical diagnosis, while TEIRESIAS tries to function as a bridge between MYCIN-- the performance program-- and the medical expert-- human expert-- who knows how to solve those problems.

Between the three entities there-- the performance program, TEIRESIAS as the bridge, and the expert physician-- we try and set up a certain sort of information flow that makes it possible for the expert to take the knowledge that's in his head, the knowledge that makes him an expert, transfer that into MYCIN the performance program. And that's the mode which we're going to look at learning in this particular example.

Now let's take a look at a slide that shows you that in a little bit more detail. And see how, first of all, this sort of thing used to be done in order to appreciate the kinds of progress we've made by putting together systems like TEIRESIAS.

This slide shows you a sort of traditional example of what learning might be like, and what knowledge base construction-- knowledge base construction, how it was done. You get together three things-- a prototype system, some programmer, typically a graduate student who's ambitious, and a domain expert-- in our case, a doctor, but more generally, somebody who simply knows how to solve problems in a particular domain. You let the three of them go to work. You hand the prototype system a particular case to work on, let it do its best on that case.

Typically what happens, of course, is it will do some sort of credible job, but it will make a mistake. So our volunteer programmer here tears off that result, hands it to the domain expert and says, what do you think? The response typically is, well, that's not bad for a start, but really what we want to do is something a little bit different. The program should really have known something slightly different there. Let me describe that to you. He describes it to the programmer, who then has to figure out what changes he has to make to that system in order to improve its performance.

So what we get is a kind of running back and forth operation here, which is tiresome for the guy in the middle and isn't even all that interesting or rewarding to the domain expert, because all he's seeing is intermittent behavior from the system.

So the question is, can we do a little bit better than that? Can we establish a more direct kind of communication, a more direct link between the expert on the one hand and the performance program on the other, so the two of them can communicate somewhat more directly? That sort of more direct link is illustrated in this next slide here.

What we have is the system on one hand, the performance system, and the domain expert on the other. And all I've tried to do is suggest the different kinds of information that might flow back and forth between these two entities. Information flowing from right to left I've labeled 'explanation.' We saw examples of that when we saw MYCIN in action because the performance program, the system here, was able to explain to the user what it is it's doing and why.

Now we have a new kind of information flow moving from left to right here when the domain expert tries to tell the system something new. We're calling that knowledge transfer. What he's trying to do is explain to the program some new piece of knowledge that it should have known, some additional piece of information that would have allowed it to solve the problem that it's trying to solve somewhat better.

Now let's zero in yet a little bit further into this kind of interaction. Let's take a closer look first at what the system looks like. And here we have as usual my favorite picture of what a knowledge based system might look like. They have this simple architecture, first. As always, the inference engine and the knowledge base. That's our performance program now.

And let me add to that the idea that we've got TEIRESIAS surrounding that, in effect, providing the link between the performance program here and the expert, who's out here. Once again we have the information flow from the system to the expert being labeled 'explanation' and the information flow from the expert on into the knowledge base being labeled 'knowledge acquisition.'

So the basic idea is that TEIRESIAS is going to function as this kind of link. It's going to be a link between the expert on the one hand and the performance program on the other. It's going to try and make it as easy as possible for the expert to say what's on his mind, that is, to say what it is that makes him an expert and allows him to solve problems in the domain that perhaps the performance program is not yet capable of doing. We want to be able to transfer that knowledge that's in the expert's head into the knowledge base of the program to improve its performance.

Now that's the model of the interaction we want. Let me show you some of the problems that are going to come up in trying to do that sort of thing. They're suggested here.

And one of those problems is what we call the incremental approach to competence. The basic idea here is that these programs take quite a while to build. As I was saying in some of the introductory lectures, we can be working for quite a while to build up these knowledge bases.

Now if that's true, if we're going to be working for a while building up these knowledge bases, then the system has to be very flexible. It has to make it easy to make those kinds of changes, and I mentioned the importance of flexibility. Well, for knowledge acquisition this incremental approach has an important aspect, too. What we mean is that we can't build the knowledge base overnight.

We can't simply expect the expert to say, here's everything I know about the domain. All you have to do is write all this down, put it in the knowledge base, and it will be done. Instead, we get this kind of incremental performance, meaning that we get a little bit of a stab at the right kind of information. We try that out. We discover shortcomings in that. We make incremental changes to it, incremental improvements to it, and the system gets better and better over time.

But the important point is we're working constantly, constantly building up this knowledge base, and constantly making changes to it. So that the knowledge acquisition process is something that's ongoing and is, as a result, going to have to be something this is as easy as possible to do, simply because we'll be doing it quite a lot.

Second basic problem we're going to run in to is this issue of formalizing what is typically previously highly informal knowledge. Now what I mean by that is that the expert is not typically used to trying to capture his knowledge in the form of simple if-then rules that we've seen before. Much more traditionally, what you have is the doctor capable of saying, well, how do I solve these problems? I think I can give you a little bit of an idea about that. Here's the kinds of things I reason about.

At best he gives you a highly informal, highly inexact impression of the kind of knowledge we're dealing with. Turning it into those nice crisp if-then rules is a challenging sort of problem. And that simply means that we want to make it possible to do that as easily as possible. We want the knowledge acquisition system to give us some assistance in trying to take somewhat ill-formed ideas and turn them into somewhat more precisely formed ideas.

And in the simplest case, all that means is it ought to be easy to take several stabs at trying to get a rule phrased just right. It ought to be easy to try it again a few times until we get the kind of thing that satisfies the expert. That is, that seems to the expert to be the kind of thing he wants to say, and that also satisfies the program in the sense that it adds into the knowledge base appropriately and improves the performance. And we'll see both of those kinds of issues, both of those kinds of capabilities, demonstrated as we go on further in the examples.

Now here are the problems-- this issue of incremental approach to competence and informal knowledge. What are we going to do? What's the solution? Well, the main idea that TEIRESIAS works on is suggested here. It's this notion of interactive transfer of expertise. What do we mean by that? Well, it's quite a mouthful, so let's try breaking down the phrase.

First of all, we're talking about expertise, because that's after all the name of the game here. What we're after is the knowledge that's in the head of the expert.

Transfer of expertise. Well, that comes back to the fact that we believe one of the best ways-- or at least one of the most effective ways-- of getting that information is to transfer it, as I've been saying, from the head of the expert into the knowledge base of the program.

And interactive. Well, that just means that we think we can best do this transfer of expertise interactively.

So in sum, we're talking about interactive transfer of expertise, which is just a slightly more detailed phrase for the model I was giving you earlier of finding a way that we can establish a link between the expert who knows the domain very well and TEIRESIAS and the performance program. We want to establish that link that lets the expert in effect transfer his knowledge into the knowledge base of the performance program, and TEIRESIAS is going to be that vehicle that establishes the link via this notion of interactive transfer of expertise.

Now as a kind of preview of what's coming up in the example that we're going to work through, let me show you some of the basic ideas that TEIRESIAS is going to use. First of all, we have here the notion of errors motivating learning. What I mean by that is it turns out, as we'll see, to be very effective to use a mistake that the performance program makes as a way of motivating the acquisition of new knowledge.

What we're saying is, let's put the performance program to work on an interesting example. Let's see how it works. Let's see in particular if it makes a mistake, because that can be very revealing. Consider what we can now do. We've got the performance program working. We've got a specific example it's trying to solve. In the eyes of the expert, it makes a mistake. And look at the question we can now ask.

We can say to the expert, here is the performance program. Here's everything it knew about this particular problem. Here's the answer it came up with. You disagree with that answer. Here's the answer you got. There's clearly a disparity here. What is it that you knew that the performance program didn't, that made it possible for you not to make that mistake on this problem?

Notice how well focused that question is. Notice that we've zeroed in on a particular problem on a particular example and are able to ask a very well specified question that says there must be some difference in the performance here. What is it exactly that you know and the program doesn't? Consider in particular how much better focused a question that is than simply sitting down with the expert and saying, all right, tell me everything you know about medicine. Or tell me everything you know about chess. Or tell me everything you know about whatever domain we're concerned about.

So setting knowledge acquisition in the context of a particular mistake on a particular problem is a very effective way of focusing the whole dialogue. And we'll see that TEIRESIAS uses this to great advantage.

Second issue, the notion of model-based understanding and of giving programs a model of their knowledge. This is best illustrated by the example that we're going to work through, but the important point is that the combination of TEIRESIAS and the performance program is in effect a system that has a model of its own knowledge, and it uses that model as a way of acquiring new knowledge. It uses that model as a way of trying to understand what the expert is trying to tell it as he tries to add new knowledge to the system.

And final point, we will see that those models that the system uses are in fact formed by experience. That is, instead of being hardwired in to the program ahead of time, those models are interactively created and improved as a result of the system's experience in interacting with the expert.

Now the best way to see all these ideas illustrated is to work through an example with TEIRESIAS, so let's try that. Let's take a few slides and see how it is that it works. Before we jump to those slides, though, let me give you a little bit of background to explain the example we're going to be working with.

As I said, TEIRESIAS and MYCIN were designed at roughly the same time and designed to work together. So in fact TEIRESIAS's first real job was to help build up the knowledge base of a medical consultation program. Now because those dialogues tend to be a little bit confusing-- with the technical jargon that's in them, they're not all that comprehensible to a wider audience-- what I've done is taken one of those interactions between the expert, TEIRESIAS, and MYCIN and translated the domain.

Where before we were talking about medical diagnosis and consulting for medicine, we've now translated things so we're talking about investment consultations. We've gone into the stock selection game here, and we've done it with a really very simple and virtually trivial mechanism. We've just done a few word substitutions. For instance, what was previously referred to in the medical domain as an infection is now referred to as an investment. We just substituted the word 'investment' for 'infection.' What used to be E. coli is now AT&T. And lo and behold, a dialogue that used to be concerning itself with medicine is now talking about stock selection and investments.

I stress that this was done by a few simple word substitutions. This is not a real operating system, but nevertheless the result is a good deal more comprehensible if we have this other domain at hand.

So let's see how all this works. Let me show you first a typical rule from that domain. You'll recognize quite readily the form of that rule because, as before, it's a simple if-then rule of the sort that we saw in the MYCIN system. Now here we've got the rule with its if and then parts. In this case though we're simply dealing with a different domain so we talk about the time scale of the investment, the kind of rate of return that we want, whether or not we know the area for the investment.

Notice in this case we have a slightly curious phrase. We're saying the area for the investment is not known. That is, this clause will be true if in fact we're unable to figure out what the appropriate area for the investment is. And basically the rule is saying, gee, if you really don't know what you want to invest in, you're willing to invest for the long term, and you want a pretty good rate of return, then what the heck, AT&T seems to be a good choice.

Now as before, we're going to use those rules in a simple backward chaining fashion that MYCIN does. Let me just review that for you. We put the rules together and try and figure out-- in this case, we're trying to figure out what the appropriate stock to invest in. We collect all the rules that tell us about what we ought to invest in. One of those is Rule 27. In order to figure out whether we can use Rule 27, we have to figure out if its preconditions have been met. In order to figure out if its preconditions have been met, we collect all the rules relevant to determining the value of its first premise clause, and so forth.

So we have the same sort of backward chaining use of the rules, and all we've really done is transformed the domain that we're working in. So the control structure looks quite the same.

Now let's take a look at TEIRESIAS, take a look at an overview of how it works. Let me give you an idea of the basic phases that are involved in trying to do this knowledge acquisition process. What is it that TEIRESIAS does, and why is it that that stuff works?

There are four basic phases to the action of the program, and they are illustrated here.

The first phase in all this is debugging. Remember that I said we're going to be tracking down an error. We're going to be doing knowledge acquisition in the context of a mistake in the performance program. So we've got an error. The question is, where did it come from and why? So the first phase is simply the debugging.

Second phase. Well, we've got to fix the bug, and that means getting a new or revised rule added to the knowledge base. So we're going to ask the expert to type in a new rule. One thing we're going to have to do, obviously, is figure out what he said.

Third phase is what we refer to as second guessing. We'll see that what TEIRESIAS does is look at the new knowledge that's going to be added to the system, the new or revised knowledge, and in effect compare it to what it already knows about that kind of problem, asking itself, gee, does this new knowledge fit in well with the things I already know about that topic? If there's a disparity, it will comment on that disparity. We'll see an example of that later on.

And then finally, of course, we need to see whether it does indeed fix the problem. So the last stage in all this is to add the new rule to the knowledge base, see if indeed it fixes the problem, if the original bug that we encountered here goes away because we've added this new rule to the knowledge base.

Now let's take a look at the system actually in operation. And as before, we have the simple sort of interactive dialogue where the performance program is going to be interviewing the user about the case at hand. Now in this case we have one small difference, which is we're going to be having the expert deal with the system.

So imagine, if you will, an expert investment consultant sitting down with the system, in effect saying aha, let's see if I can challenge the system. Let me make up a case. Let me see how well that program performs on this case. And then let's see if it makes a mistake, whether I can add some new knowledge to fix that problem. So here's an example of the expert interacting with the system.

And as before, what we see is the program in effect interviewing the user about this particular case, asking in this case name, age, income, number of dependents-- the sorts of things that would be relevant to trying to determine what an appropriate investment would be for somebody with this kind of occupation, income, and et cetera. And like the MYCIN system, eventually it comes up with a conclusion.

That is, having interviewed the user about this particular case, asking questions about what it is that this family situation is like to determine what the appropriate investment would be, eventually we come up with a conclusion. And those are shown on this next slide.

Here are the conclusions that the system suggests. In this case, it turns out that it looks like an investment in Varian and AT&T would be a good choice for this particular investor. Up to now, up to this point right here, we've been dealing with the standard sort of interaction between the performance program-- in this case, the investment consultant-- and the user.

But now because indeed what we really have is the expert challenging the performance program, TEIRESIAS now stands in between those two, interrupts the flow of information back and forth and says, wait a minute, are these answers correct? Is the performance program's answers the appropriate ones in this case? And now we've got the expert responding in the role of the expert and saying, aha, no, there's a mistake here. So we've got our bug, and now the problem is to go track down the bug.

Well the first thing to do, clearly, is to get that bug better specified. All we know so far is that the performance program's answers are incorrect. So the first thing to do is find out in a little bit more detail what it is that's wrong. That's illustrated here.

We ask a couple of obvious questions. Like let's see-- if the suggestions that the performance program made are incorrect, one question is, were there any suggestions missing? That is, did we leave out some stocks that we should have recommended? And in this case, indeed, the expert feels that that's true-- there are some recommendations which have been missed. In this case Digital Equipment, Data General, should have been suggested by the program but were not.

Then the obvious converse question, were there any that showed up and shouldn't have? And in this case, the expert feels that AT&T was an inappropriate choice. The system should not have made that recommendation.

Now there's a simple heuristic built into TEIRESIAS that says if there's a recommendation that showed up inappropriately, that's probably a pretty good place to start tracking down the source of the error. That is, let's start there with this thing that shouldn't have appeared and say, why did it appear, and how can we make it disappear?

Now the kinds of dialogues we get into at this point are fairly extended and fairly detailed, so what I'm going to do from now on is show you snapshots out of the middle of the dialogue that's going on between the expert and TEIRESIAS to try and debug the program. Trying first to track down the source of the error, input the rule, understand the rule, and so forth. I'll be giving you just small snapshots out of that dialogue in order that we can move through it without getting too bogged down in detail.

The first problem, as I said, is tracking down the error. What happens here is, as I said, we start with this suggestion that shouldn't have appeared-- the AT&T suggestion-- and TEIRESIAS asks the obvious question. It looks back into the performance program, discovers why the AT&T suggestion was made in the first place, picks up that rule and says, aha, was this rule correct? Because one clear source of error is the rule that gave us the AT&T to begin with might itself be wrong.

We simply ask the expert to comment on that. We show him the rule and say, was this rule correct? If that rule is correct as it stands, then it must be that that rule should never have been invoked. Which means we can move one step back further in the process and say, now gee, one of the premise clauses for that rule should have been false. It turned out to be true in this situation, so there must be a problem further back in the chain. We work our way methodically back through the chain of reasoning that led to the AT&T.

Now that's an important point, because look what it gives us. It gives us a very methodical way of figuring out where it went wrong. We're not simply asking the expert to comment on the program's performance. We're not saying, OK, what do you think? And getting back a vague answer like well, it's not bad, but it lacks a real feeling for this domain.

Instead, we're asking some very specific questions. Is this first rule correct? If that rule is correct, should one of its premise clauses have failed? If so, then one of the rules that made that premise clause true must have either itself been in error or been invoked incorrectly. So we have a very methodical way of asking very sharply defined questions to which the expert can only respond yes or no, there's something wrong here. Or yes indeed, that rule is correct, and we can look further back on the chain.

Now as a result of that dialogue, we eventually tracked down the error in this case. Let me show you that on the next slide. This is a summary of the problem as we finally discovered it. It's shown here.

And what we're going to do is start off with the AT&T way off over here at the right and simulate for you the kinds of questions that go on, simulate the process that helps to track down the error. Recall that AT&T shouldn't have appeared. So the first question we ask is, well rule 84, as it turns out, is the one that gave us the AT&T. Is that rule correct? The expert says yes indeed, the rule is sound in and of itself.

Well then, moving back further. Aha, there must've been a problem here. Each of the premise clauses evaluated to true. One of them should not have. And the expert says yes, that's right. One of them shouldn't have. In particular the area, the clause about area, should not have evaluated to true. Now remember that this one is a little tricky. That clause says, if the area of the investment is not known, and in this case the area of the investment wasn't known, so this clause could be true, so this rule could fire.

So we have to back up a little further and say, hey, why didn't we know what the investment area should have been? Backing up a little further, the system is able to look up the performance program's record of its performance and say, aha, the only way we could have figured out the area for the investment was with either of these two rules. Are they correct as they stand? And the expert says, yes indeed, those rules are correct as they stand.

We back up yet further and say, aha, look here's the reason they both failed. Rule 116 failed because its third clause was false. And as it turns out, Rule 50 also failed because of its third clause, a slightly different one, was also false. So putting this all together again, let's see. Rule 50, Rule 116 both failed. That left us without a value for area. Since the area was not known, this clause was true. This rule succeeded, and the AT&T appeared.

So what's clearly going on is-- backing up here to this point one more time-- what's going on is that we should have known a value for area. The two rules that might have given it to us failed. So the system's conclusion is that aha, there must've been a rule missing. The system must be lacking a rule that would, so to speak, fit in down here, tell us something about the area for the investment. We'd know what that was. This clause would be false. Rule 184 would fail. And finally, the AT&T would go away.

So notice what we've done. We backtracked, as I said, through that sequence of logic that led to the error, that led to the stock recommendation we didn't want. And we did that via a very well focused sequence of questions that makes the expert respond rather precisely, helping us track down the source of the error. In this case what it looks like is we're missing a rule.

Now another thing that comes out of that dialogue is a very useful form of focusing for the program, as well. I've been saying that the dialogue is well focused, that forces the expert to answer some very specific questions about whether certain rules are right or wrong. But it turns out the program makes very good use out of that focusing, as well. In order to show you how that works, I have to explain one more idea that's in the program that gives it a lot of its power. That's the notion of a rule model. The overall idea about a rule model is illustrated here on this next slide.

The basic idea is that we have an abstract description of a subset of the rules. And that abstract description is formed from a number of empirical generalizations about those rules. Now what do I mean by that? Well, I mean the following. If I take a look at all of the rules in the system that tell me something about, let's say, what area to invest in, there might be a half dozen or a dozen such rules. And then it turns out that I will notice certain regularities in those rules.

I might notice, for instance, that the kinds of concepts that show up in the premise clauses of those rules tend to be fairly repetitive. That is, most of the rules in that particular subset tend to mention the same sorts of things in their premise clauses. And that's an interesting and useful kind of regularity. And that's what the rule model tells me. It tells me about the kinds of empirical regularities which show up in the structure and content of the rules in the knowledge base. So let me show you that in a little bit more detail. That's on the next slide here.

Here's a little bit more detail about what a rule model looks like. We have the notion of first of all, the examples. That is, the examples from which this particular rule model is constructed, the subset of rules which this model describes. And then we need the important part that is the description. That is, tell us something about this subset of rules that will be useful. As is suggested here, what we want to do is somehow characterize a typical member of the subset.

What does that mean? Well, in this case since we're dealing with if-then rules, we'd like to characterize the premise and characterize the action. And what do I mean by characterization? Well, I mean first of all which attributes or concepts typically appear in a premise of a rule. As I was saying before, there are certain sorts of regularities that appear in the premises of the rule, certain concepts that show up repeatedly.

There are also correlations of attributes or correlations of concepts. That is, if I typically see one idea in the premise of one of these rules, there may be another idea that also typically shows up. So sometimes these things come together in pairs or triples.

So that's what I mean by a description. That is, in the most general terms, I'm trying to characterize a typical member of this subset. And in more detail, I do that by talking about what kinds of ideas typically appear in the premise of one of these rules, and of correlations of ideas-- ideas that seem to show up together in the premises of these rules.

Finally, it turns out that the rule models are also structured into a generalization hierarchy. That is, some of these models are more general than others, some more specific. And I simply have the appropriate sorts of structure here which point to models more general than the one I have here and some that are more specific. It turns out that the system is capable of picking out the appropriate sort of model on this hierarchy in order to do its work.

Now, how does all that get used? Well, in order to see that, let me show you one more slide out of the dialogue of the expert interacting with the system in order to add a new rule to it. As a result of that methodical tracking down of the error, the system is capable of saying, aha, here's the problem. In fact, here's the source of the bug that we've got here.

Notice what I said before-- there seems to be a rule missing, a rule that would give us an area for the investment. In fact, we track down even a little bit further and go ahead and ask the user what the area for the investment should have been, in order that we can be fairly precise. And that's shown here on this next slide.

That is, the system says first, summing up, I need a rule that would allow me to deduce that we want to be in the high technology area. Do you want to give me a rule like that now? And the user says, yes indeed, I will. And then proceeds to type in the appropriate text. Here's the user's natural language version of the rule he wants to add to the system. It says in this case if you're in the 50% income tax bracket and you're somebody who closely follows the market, then that's evidence that maybe the high technology area is a good place for you to invest in.

So in effect, what we're saying is if you have a lot of money and you're willing to be careful about how it's invested, spend some time following the market, then maybe high technology is a good place to go looking for the appropriate sorts of investments for you.

Now how does all this come together in an attempt to make some sense out of this rule? Notice what we've got. We've got that natural language text which we want to interpret. Now by interpret I mean we want to take that English language, turn it into the internal representation-- and in this case the internal representation is a Lisp representation of that rule-- and then what we're going to do is turn that back into English and display that to the expert and say, is this what you meant? And let him comment on that. The question is, how do we do that? This is that next phase of the process, which is interpreting the new rule.

Now as many of you may know, natural language understanding is all by itself a very difficult problem, a research topic in and of itself, and it was not the focus of this particular research. So instead, rather than trying to confront this as a full fledged natural language understanding issue, a number of considerably simpler mechanisms were developed. So one reason for not trying to be sophisticated about natural language is that it's a hard problem in and of itself.

A second reason is that it turns out we can take advantage of the technical language that's being used here. Not surprisingly, there are technical terms in any particular domain. Simply by being willing to spot giveaway keywords in the language, we can get fairly far with understanding what is being said.

There's yet a third reason for not trying to be too clever about the natural language processing here, and that's that the way the experts phrase the rules are not always entirely grammatical. They use the jargon. They use technical shorthand. There's usually a great deal of information compressed into just a few words, and those few words do not always follow the normal everyday rules of English grammar.

So there's several reasons why we don't necessarily want to take a traditional approach to trying to do sophisticated natural language, but instead can rely on some somewhat more simple mechanisms, but mechanisms that nevertheless do the job. To show you the general idea behind TEIRESIAS's approach to this problem, let me show you a kind of schematic of the way it works.

The basic idea is illustrated here. There are two basic components to the system's attempt to understand the rule. We have what's called expectations that help with the function in a goal driven manner, and the signal itself-- that is, the text-- that helps it to function in a data driven manner. Let me explain a little bit about both of those.

Remember all that tracking down we did. Remember the effort we went to to carefully try and figure out where the bug was, that methodical backtracking through the sequence of rules that were invoked in order to figure out what was wrong. What that eventually led us to was the fact that we had a rule missing from the system, and indeed a rule that had to do with investment area. As a result of that, the system is able to say, aha, I expect a rule that's going to tell me something about what the investment area is supposed to be. That is, it can set up certain expectations.

How are those expectations in fact manifested? Well, it turns out that the system picks out that rule model which describes the subset of rules dealing with investment area. So as a result of tracking down the source of the error, what it's able to do is pick out a particular rule model and in effect say, aha, I expect a rule that looks like the kind of rule typically found in the subset. That's what allows it to do a certain amount of goal driven processing. That is, because of the rule model, it has certain expectations about what it's going to find in the text, and it can go looking for those things.

Remember that we also have the signal itself, that is, the text itself in here. And we can do the kind of keyword spotting I was talking about before. And that we refer to as a data driven process because we're taking a look at the kind of information that we've got, looking at the words that were used, and seeing what that suggests about what the expert might be saying.

And these two processes are in effect intersected. I ask myself, in effect, what did I expect to find? Did I find any of those concepts here? What keywords did I actually see in the text? What did they suggest that I might actually be seeing there? Let me put those two together and see if, in fact I can't come up with a reasonable guess about what the expert said in this rule as a result of those two processes.

Now not surprisingly, that's useful but imperfect. Its main virtue is that there's not a tremendous amount of overhead there in terms of trying to do sophisticated natural language processing. It also has the virtue of taking advantage of the context that we've been able to set up, in particular that context that says having tracked down the source of the error, we know a fair amount about what it will take to fix that error. We make use of that by picking out a specific rule model, and then can use that in this kind of intersection process to produce our result.

Now as I said, it's useful but certainly not a foolproof process. So one of the other things that TEIRESIAS has is a very simple sort of editor. That is, it has a little mechanism that allows the expert to review the program's first guesses as to what it is the expert has said, the program's first attempts to translate that rule into Lisp and then back into English, and allows him to make some changes-- to delete spurious clauses that may have been introduced, or to say, no, you didn't quite get that clause right. Let me try and rephrase that one part of the rule. And the system will make an attempt to understand. Let me show you how that works.

Again a segment out of the middle of the dialogue. Here's an example of the system asking, well let's see. We've been making some changes here. Are there any other changes you'd like to make? And the expert simply types in a question mark which says, in effect, well, show me what I've got so far. And what we've got so far is the premise of the rule. We've managed to get the first clause and the second clause correct. Apparently the system also introduced a third clause that was spurious here, and the expert in some earlier editing said, no, get rid of that. Let's delete that one. The system then says, is there anything else you'd like to change? And the expert says, no.

So look what we've got here. We've got a little editor that allows the expert to say, well, let's say you misinterpreted a little bit of what I had to say. Let me change this a little bit, make those revisions to it. And now he's happy with the premise of the rule. So let's go back and see what we've got in the conclusion. Do you want to make any changes to the action? Turns out in this case the answer's no, we don't need to make any changes. As it turns out in this case, the system was able to get the action part of the rule, the then part of the rule, figurec out correctly the first time around.

So since we seem to be finished here as far as the expert's concerned, the system reviews it and says, all right, here's my current understanding of your rule. If we're in the 50% income tax bracket and this guy follows the market carefully, we've got some evidence that it should be a high technology investment. Is that OK? The expert says, yes, indeed.

Let's consider what's going on here. Yes, indeed, the expert and the program are now together. That is, the expert believes the program does indeed understand what he said, it does indeed understand the rule that he's given it. But now we have this third phase of the knowledge acquisition process that I was talking about before-- the notion of second guessing.

What the program is going to try and do is take a look at this new rule and in effect say to itself, let me compare this new rule to my existing set of rules that look like this. And let me see whether this new kind of information seems to fit in with the existing stuff that I've got. Does it look like what I expected it to look like? And if not, can I comment appropriately on the disparities, on the interesting and important differences between this rule the expert just gave me and my idea of a typical rule out of this subset of rules?

So you can see what I'm leading up to. What the system does is look at this rule and compare it to the rule model that characterizes rules in that subset and say in effect, does this rule look like those others? And if it doesn't, can I say something intelligent about the disparity that shows up? Let me show you what happens when in this case, the system compares its answer to the typical rules from this subset. And that's illustrated here.

The system comments that, let's see, I hate to criticize. I hate to criticize, but did you know that most rules out of this subset, most rules that look like this that mention what the investment might be-- rules about investment area-- that mention first of all income tax bracket and second of all how closely the client follows the market, rules like that typically also mention the amount of investment experience.

So notice what we're doing here. We're looking at the kind of regularities that we find in the rules. And in this case one of the regularities that the system has noticed is that rules about investment area that mentioned both this thing and this thing typically also mentioned this thing, as well.

So the system has noticed a disparity. It's saying, in effect, this new kind of knowledge doesn't quite jive with my standard picture of what a rule like this looks like. Having noticed that disparity and commenting on it, it says, aha, can we do something about fixing it? And what it offers to do on behalf of the user is shown here.

Having noticed the disparity, the system is then able to say, aha, shall I try and write a clause to account for this difference? There seems to be something missing. That is, the notion of the amount of investment experience of the client seems to be missing. Shall I try and write a clause to account for that? User says, yeah, sure. Go ahead and give it a try. As it turns out in this case, the system has enough information available from the consultation we ran a little bit earlier to make a good guess about what this clause ought to be.

The simple heuristic here, by the way, is that if this rule is really going to work, this new rule is really going to work and fix the problem, then what we have to do clearly is write a clause here which has to do with the amount of investment experience, but more precisely, has to be true in the context of this particular consultation. And if you happen to remember far back enough, you'll recall that this was an investor with a moderate amount of investment experience, so this will evaluate to true, which means indeed that this new rule will still work in this context.

And the expert, as always, gets the last word here. The system says, does that look like a reasonable guess at that clause? The expert believes it, and now we've got our new rule.

So once again, the expert gave his version of the rule. We went through a little bit of understanding and debugging of what the expert actually typed in. The expert was initially happy with what the system understood. The system then compared that new rule to its existing model of what rules like that should look like, commented on the difference, augmented the rule so that that difference disappeared, and now I can add that new rule to the knowledge base.

Now we enter the last phase of the knowledge acquisition process, which is to say, now that both the expert and the system are happy with the rule, let's add it to the knowledge base of the performance program and just start that performance program up again and say, all right, try this problem over again, and let's see if we can improve the results. So let's see how that works.

Before we do that, we have one last piece of bookkeeping to attend to, and that's illustrated here. Before we can actually run the program, we just want to get a little bit of bookkeeping here.

One thing we're going to ask for is a comment from the user on why this rule was added to the system or why a rule was updated. And we simply take in the text the user has given us. He'll type out some explanation here. We make no particular attempt to be clever about parsing this, but simply store it away with the rule so that we can say, aha, this was what motivated this rule. This is why it got added.

The rule then gets added to the knowledge base, and then we go ahead and re-run the consultation.

Now in addition to that little bit of information, that sentence which gets added to the rule and stored away with it, there's a number of things that go on behind the scenes in terms of the bookkeeping there which weren't obvious from that slide, but very useful nonetheless.

We tag the rule with the date and time that it was changed or added. We tag it with the name of the author so we know who to go asking if the rule seems to make no sense later on. Remember we had that little explanation about why the rule was added. We also include a pointer to this particular consultation. All of that information becomes useful because of what I was talking about earlier-- this notion of incremental change, incremental approach to confidence.

We're going to be at this knowledge acquisition task for a long time-- perhaps several years. It's very easy, six months after we add a new rule, to go back and look at that and say, wait a minute. Where the heck did we get that? That rule makes absolutely no sense. Whoever thought of adding that to the system, and why did we put it in there to begin with? So these simple bookkeeping things can do a lot of that for us.

At the very least, they can tell us when we got the rule, who added it, and why that thing was there to begin with. We can look back at the context which motivated us to do that to begin with. And that will make life easier because we're going to be at this for a while. Just to have those pointers makes life a lot easier in terms of making sure we're not putting rules in, then taking them out again later because we forgot why we added them to begin with.

So if we develop a better understanding of the domain, maybe we don't like the rule in the first fashion that we added it to the knowledge base, but at least we understand why we added it. And if we now understand that problem better, maybe we can generalize the rule a little bit. Maybe we won't just throw it away. We'll understand why it was there to begin with, how we can generalize it, and how we can fix it so it fixes the problem even better this time.

So we've gone through this whole thing figuring out where the bug was, translating the new rule, tracking down the source of the error, using that information so we could get a better guess at what the expert's telling us, seeing if the system is happy with the rule. The last thing we do, as I was saying, is to re-run that consultation to see if it doesn't in fact absolutely fix the bug.

So here's the very last part of that consultation there. This time, as I said, TEIRESIAS is kind of poking the performance program and saying, OK, you've got an updated knowledge base now. Go back and try again. Let's try that example again. And that's shown here.

Here's the final result in that process. And this time, lo and behold, not only has AT&T disappeared, but curiously enough, the other two suggestions which were missing last time-- Data General and Digital Equipment-- have appeared. The program, TEIRESIAS, can't resist patting itself on the back. It comments that things seem to be straightened out. But as always, the expert gets the last word. Are these values indeed correct? Yes, says the expert. So, good. Indeed the problem has been fixed.

Now it's unusual that we get quite this lucky, where we managed to fix three different bugs with a single new rule. Let me just review for you what went on inside the system to show you why it is that in this case we did get lucky, and we get a rather nice increase in performance by the addition of a single new rule. Here's a schematic of how that works. Let's trace back and forth through it.

Here's an overview of what's going on. And let's start by zeroing in over here on the right at this AT&T. Remember I said we start there because AT&T shouldn't have appeared. And we backed up from there and said, well, wait a minute. Why did we have the AT&T? Well, that came up because of this Rule 27 here.

And why did that work? Well, we back up a little further and discover that that was there because we didn't have any value for the area for the investment. And earlier on we lacked a value for area because we only had these two rules-- 116 and 50-- to give us that area. They both failed. We got no value for area. That allowed Rule 27 to succeed, and that gave us the AT&T.

But now we've added this new rule. We've added 383. That was the new rule we got a few minutes ago. That gives us a value for area, and that does two nice things. It first of all inhibits Rule 27 from firing. Because remember, 27 fired because it lacked a value for area. Now we've got one. Now Rule 27 won't fire, and the AT&T will disappear.

But more than that, it turns out there was another rule in the system-- 84-- which was just sitting there waiting. It needs a value for area. It didn't have one last time. Now it does because 383 is there. And it turns out that Rule 84, in turn, is the one that gives us our new values.

As I said, we get lucky here because that single new rule was able not only to fix the problem with the fact that AT&T was appearing, but was also able to offer the value for area which triggered Rule 84, which gave us a hint about our missing values and was able to supply those.

Now as I said, of course, we don't always get this lucky, but this is a real example-- real in the sense that it was originally a medical example. It was a medical example in which one diagnosis was made incorrectly, two others were missed, and schematically, at least it was exactly this problem. I've translated it, as I said before, translated it into the investment domain. Nevertheless, the point stands that occasionally we do get this interesting sort of behavior where what looks like a complicated bug in fact comes down to missing a single rule in the system.

So, to review for a moment. TEIRESIAS works in this knowledge acquisition process by first of all tracking down the source of the error, trying to interpret the new rule that has come in, second guessing the expert on the rule trying to see if there's something missing, and finally, poking the performance program to re-run it and seeing whether or not we fixed all the problems.

Now let's go back and review some of those main ideas about how TEIRESIAS works. Let me show you again what we're talking about by some of those notions. Here's some of the ideas we were working with.

First of all, we have the notion of knowledge acquisition in context. That is, we were saying let's work on a real example and let's let the performance program do its best on a particular example and see what happens. What happens in this case was that the program made a mistake, and that provided a good deal of context, provided a good deal of information that allowed us to focus right in on that problem and gave us some hints about how to find the source of the error, how to fix it, how to interpret the new rule.

We saw the notion of model based understanding and of giving the program a model of its knowledge. In this case, the combination of TEIRESIAS and the performance program is together a system which, in a primitive sense at least, has a model of its own knowledge. In a sense, it knows what it knows, and it uses that knowledge-- the rule models in particular-- it uses those in what we call model based understanding. It tries to match what it sees in the new rule against what it expected to see as a result of having that rule model.

Fourth, we have this notion of testing new knowledge. That is, we want to test the new knowledge to see how it fits in to the existing knowledge base. This is what gave TEIRESIAS the ability in effect to second guess the expert, to say, aha, does this piece of new knowledge look like what I expected to find?

And finally, it turns out that the rule models are not static. They're not hardwired into the system. Rather they are formed and improved as a result of interacting with the expert. Now that's interesting because most systems that use this notion of model based understanding had to deal with models that were hardwired into the system. That is, they were a static set of models that were there to begin with.

And if we're going to do understanding as a process of comparing a model-- in this case, a rule model-- against the kind of things we're seeing in the world, then the set of rule models is the sum total of what we're able to understand. The set of models constrains how much we know about the world. So it's interesting, therefore, to be able to say that the system is capable not only of updating and improving those models, but in fact of forming new ones as a result of interacting with the expert. Let me show you how that works.

That's suggested here in this diagram that shows you some of the information flow that goes on. At the far right here we've got the expert interacting with TEIRESIAS. The result of that is this rule acquisition process. And that rule acquisition process is, as I was showing you a little earlier, aided by having around the rule models. Remember, it was having those models around, letting us get some expectations from them, that helped to make this rule acquisition process work well. That's what supplied the expectations and gave us some of our goal driven processing here.

But as a result of doing this, as a result of getting a new rule, we added to the knowledge base. We have added and improved our knowledge base, made it a little bit larger. As a result of that, we can go ahead and update the rule models, because remember what I said. I said that these are empirical generalizations about subsets of information in here. Well, the larger my subset, the more likely my empirical generalizations are to be valid. The larger the number of data points I have to generalize on, the more likely it is that my generalizations will be good.

So each time I make a change to the knowledge base, I go ahead and update the rule models. Having updated them, it's very likely that my performance on knowledge acquisition, on rule acquisition, the next time around will actually get better.

So notice what's going on here. Not only do we have the models, but because we have this ability to update, improve, and even create new models interactively, the system is able to get better in its knowledge acquisition capabilities. Simply because it has a slightly better picture of its own knowledge base-- the knowledge base is now larger-- our empirical generalizations are likely to be better. As a result, that kind of information then can feed back into this process of doing knowledge acquisition.

So let me summarize by going back to these points here.

We have the notion of knowledge acquisition in context. We want to be able to get knowledge by looking at a particular error.

Model based understanding and giving a program a model of its knowledge. In a very primitive sense, the system knows what it knows.

We test the new knowledge to see if it fits in with the old.

And an interesting capability-- the system is capable of forming and of updating the models that it has, forming and updating models of its own knowledge, as a result of experience in interacting with the expert. Thank you.