2-15-1985

Unification in Linguistic Computing

Martin Kay

Follow this and additional works at: https://scholarsarchive.byu.edu/dlls

BYU ScholarsArchive Citation
Available at: https://scholarsarchive.byu.edu/dlls/vol11/iss1/3
It is really delightful to be here. I have known people at this university for as long as I have known anything, and that's at least three weeks. And it is delightful finally to be here and see some of them on their turf. You may say that it's cold outside and it's warm where I come from. It was actually only 80 degrees yesterday. But actually I feel very warm here. I'm amongst old friends from way back and it's very nice to be here.

Now I suppose I should worry about the title of my presentation because I realize that in these surroundings 'unification' could mean more than one thing. And all I have to say on that score is that I have checked all this out with the Reverend Moon and it is as it ought to be.

I am a computational linguist and I want to start out by saying just a few things about what I take computational linguistics to mean because that is the backdrop against which all this other stuff will be played out. I take computational linguistics to be a term which has two referents—it can mean two different kinds of things—one of them a big thing and the other one a smaller one included within the big thing. The big thing of course is anything that you care to do with computers that has something to do with language. And a lot of that turns out to be engineering of one sort or another. That is to say, you have something that you want computers to do and you want them to do it with natural language because either the natural language existed before you thought of bringing the computer along or because there are people there that want to deal with the computer and are not prepared to do so in the languages it naturally speaks. You are therefore going to have to persuade them to do it in languages like English, for example. So under this broad heading of computational linguistics come obvious things like machine translation, natural language access to data bases so people can have questions answered about it, and so forth and so on. The use of computers in language instruction, I suppose, would fall under that heading. There is almost an indefinite list of things that you can add to it. Let's call it 'computational linguistic engineering' for want of a better term. That is all part of computational linguistics in the large.

Then I also like to think of computational linguistics as having a much narrower meaning and it is in this narrower meaning that I pursue it
most of the time. In that case it is not an engineering enterprise so much as a scientific enterprise and its aims are exactly the same as the aims of theoretical linguistics in general: namely, to understand how in the world it can possibly be that by blurting out these noises at one another that we come to have different ideas than we had before the exercise started--how communication, in other words, can be brought about by this really remarkably narrow channel that we manage to establish between one another. Some people are not even amazed by this. But I think if you think about it for a little while and think of the things that you are indeed able to do by blurting noises at other people you will see that it is indeed a scientifically puzzling thing.

Now computational linguists simply bring a new perspective to this same problem that everybody else is trying to answer. They are not trying to do some different thing with it; they are trying to answer the same questions. And in this narrow sense of computational linguistics I take it that they don't actually need to use a computer in principle at all, though they usually in fact do. What they do need to use is their knowledge of how computers work, their knowledge of computer science, their knowledge of symbol processing, their knowledge of certain kinds of logic which turn out to be appropriate inside computers, their knowledge of data structures, and their knowledge of things of that sort. That brings a very different perspective to the kinds of things that you might do as a linguist. This is not surprising if you believe, as I do, that about the only other thing there is around on this planet that performs operations on symbols--properly understood, things which both have a form and a meaning--the only thing that performs operations on symbols in a serious way other than a human being is a computer. Now it may not do it in the same way and it may not be sophisticated in certain ways and it may not have quite that same appreciation for some of the subtleties of poetry that human beings do, but nevertheless it does process symbols. Furthermore, we have some considerable control over the way that it processes symbols. And so it is just possible that from these various ways that are open to a computer scientist we could get a certain amount of inspiration, let us say, about how linguistics might be done. And the people that search for that inspiration in computer science are, to that extent, computational linguists, and some even think they have found it, I among them.

Now there are several questions that you might ask at this point: if there have been a number of computational linguists around doing this sort of thing for a long time, then why haven't they had an impact that I could have noticed on the rest of linguistics, and that is a fair question. And I think there is a good reason why it has taken so long for the impact to be felt but I think it is now being felt. And the reason is that it has taken us this long to understand some of the more fundamental things about computer science in general, and we had to understand those before we could start trying to sell them to somebody else, such as linguistics.

The most interesting thing that I have to say to you today--I hope I can get across the feeling of this even if I cannot give you the details of it--is that noncomputationally-inspired approaches to linguistics in recent years look, on the face of it, more computational than the computationally-inspired ones look. And there is a very good reason for that and it is really quite strange but a fascinating story and it is really the story
that I want to tell. What computationalists have discovered over the years is 'unification' in one form or another and therefore it is going to be up to me to try and tell you what unification is about and why it is such a good thing.

If you look through recent work on linguistics, especially in North America, where formal linguistics is so much in vogue, you will find a lot of procedures involved in doing whatever it is that gets done. If you look in any of the multifarious and extremely challenging work that has been done in the formalist tradition since Chomsky published *Syntactic Structures* in the mid-50's, you will find everything set out with tremendous precision and you will find rules of a more exact sort than was found in linguistics before, and associated with these rules very, very carefully orchestrated sets of procedures for how they are to be applied. So, for example, according to the theory of *Syntactic Structures*, a grammar started out with rewriting rules, famous ones like:

\[ S \rightarrow \text{NP} \text{ VP} \]

The way you are supposed to generate a sentence in this is to start by writing down \( S \). Why? Well, because that is how you start. There is no other particular reason, that is just part of this formal system. Then you can take whatever you happen to have written down and you can replace it by whatever the right-hand side of the rule says. And then there are other things that say a noun phrase, for example, can consist of a determiner plus a noun:

\[ \text{NP} \rightarrow \text{Det} \ \text{N} \]

If you hook these up with lines, which is the traditional thing to do, you get tree-structures out of all this, that you have all seen too many times to care to remember.

So that was the first thing that happened in the generation of a sentence. There was no claim that this went on in people's heads when they were generating sentences; this does not claim to be a psychological model of the steps people go through. This is simply a grammar which is sufficiently precise so that we know for any proposed sentence whether it really is a sentence of the language or not, and secondly what its structure is and therefore how one would go about figuring out what it means.
So you started out by building a tree like this, and then you applied to it a set of transformations, and these transformations take a tree into another tree, and into another tree, and so on, until there are no transformations left on the list, in which case the tree is what is called a 'surface structure'. If you want to know what the sentence really sounds like, then you read off all the words from the bottom of it.

If you are not familiar with all that, it really doesn't matter. Details are not important. The important thing is that you must go through these steps in just the right order. Each of these transformational rules can in general work a massive change on the tree; it may look very different from the one before. The rules are ordered in a list—Transformation 1 through Transformation n—and you must take them exactly in the order given and some of them are marked as obligatory and some of them are marked as optional and by golly you'd better not miss doing any of the obligatory ones and stuff like that, otherwise you'll get the wrong answer.

So there is a very important sequence of steps that gets carried out. It occurs everywhere. It occurs in phonology—let me translate phonology into graphology, which, when you see it, looks a lot simpler. For example, there is a rule in English that says [there is] some funny kind of S—which nobody has ever seen but we can think of as existing in the head and which comes out as the plural on most regular nouns in English and as the 3rd person singular on verbs—that gets rewritten as es in a situation following a number of things, amongst them -s, -sh, -ch, -x, -z, and -j and things like that; otherwise, there is another rule that says S gets rewritten as s:

\[
(1) \quad S \rightarrow es/ \begin{cases} s \\ sh \\ x \\ \text{etc.} \end{cases} \\
(2) \quad S \rightarrow s
\]

It is obviously important that the rules should be ordered in this way, because if you were to carry the second rule out first there would be no big Ss left for the first one to work on at all and the words that ended in these magic things would have just a bare s after them and not an es. So you have to get the order of the rules right. You must, in other words, do a certain sequence of steps in a very certain definite order. Transformations were like that; early computational linguistics was like that.

There were types of grammars in early computational linguistics called 'augmented transition networks', and I will give you a sort of comic book view of what those are, the same sort of comic book variety I gave you of the other things. But each one of these is a semester course, you know, and I've got to pretend that I really know what's in it.

In this you imagine machines that can be in a certain state at a given moment and at the next moment the machine can move to another state, and given that it is in a particular state, perhaps of a variety of different states that it could go to, we assume that at a given moment it is sitting
looking at a particular word in the sentence:

\[
\begin{array}{c}
S_1 \quad \longrightarrow \\
S_2 \quad \longrightarrow \\
S_3
\end{array}
\]

And the state it goes to next is determined by the label that we have on the lines, and by that word in the sentence. Some words will allow you to go one way, some words will allow you to go another way, and so forth. So, for example, right at the beginning of the sentence you might do something that says, "look for a noun phrase," and if you find a noun phrase, call it the subject of the sentence.

\[
\begin{array}{c}
NP \\
\text{Subject} \quad \longrightarrow \\
S_1 \quad \longrightarrow \\
S_2 \quad \longrightarrow \\
S_3
\end{array}
\]

And then you trundle on through some more of these states here and then over on the right you find a verb which has passive morphology—it is a passive verb. And then you say, "Oh boy, we blew this one didn't we? This wasn't the subject after all. At least not for the answer we would like to get out. We'd really like to fix up passive sentences so that what we learn about them makes them look very like active sentences but we blew it though at the beginning here because we didn't know that the verb had passive morphology." That's all right, don't worry about a thing. What we will do is take what used to be the subject and put it in the object pigeon-hole and we will clear out the subject pigeon-hole and go on as though nothing happened.

That was wonderful, and they parsed a lot of sentences with this grammar, a lot of sentences about moon rocks and wonderful things like that a long time ago, a part of the folklore of our field, this. Now, the trouble about this is you must start at the beginning and work your way towards the end. Suppose that you were taking the description of a sentence that this gives you, and you were trying to get the sentence from it. Notice that you could not reverse the procedure. This thing will allow you to analyze sentences and it is absolutely hopeless if you try to use it for generating sentences. What would you do? You would say, "Ah, well the first thing I've got to do is to take the subject and put it down here at the beginning of the sentence as a noun phrase." So you do that. There it is, it is down in the sentence and you have moved on. And when you come over to the object, you decide that you need to put out a passive verb and you do that, and furthermore you infer into that that everything you have done up to now was wrong. And there is nothing like the above that you can do that swaps it around because that has already been said. It is all over.
This is standard linguistics. Look, there is nothing wrong with this. The only thing I am trying to point out about it is that it is full of computer procedures, full of programs—that is what is remarkable about it—more full of programs than you will find in any of the things on which it was originally based, and by which it was inspired.

Let me just digress for a little moment about that. Notice how most of the styles of linguistics that we have seen have been inspired by some other scientific enterprise that just happens to be around and popular at the moment. Early linguistics in Germany—the traditional historical linguistics that is still pursued—was a genetic model of language. Let us see what we can do if we treat languages as belonging to genetic families. And we looked into that for a while and we discovered that, by golly, it works. This is a gold mine of interesting stuff.

In American structuralism, we took a chemical model...why not put it that way? What we were trying to do was to find what all the atoms were, find out what the elements were, what are the different classes of things which go in here. So we worried about whether, in 'cranberry', 'cran' is a morpheme or not, because after all, 'blueberry' is a berry and it's blue, but 'cranberry' isn't a berry and it's 'cran'. You worried about whether 'cran' was a morpheme, you worried what that element was. It had to fit into the periodic table somewhere.

Transformational grammar is inspired by logic. Logic tells you how, given one well-formed formula to get another well-formed formula from it by rewriting it. But the operations that you perform in logic are, for the most part, extremely simple and they have the property that if there is a variable, that either that variable has a value, it is bound to a value and it is bound to it for all time and that is the value of that variable, or it is a free variable, and remains unbound. And if you work your way through a proof or a computation or a logical derivation of some sort, you may discover what the value of the variable is, whereas you did not know that before, but you can never change it from what is was to something else. Never in logic do you pull a number like changing the values of variables. You don't say, "Well, I'm sorry. I didn't mean that about 'x'."

Unification, very broadly speaking, is computing with variables whose values you may come to know whereas you did not know them before, but once you know them, they never change. That is what unification is all about. (I checked it with the Reverend Moon, like I said in the brief.) The interesting thing is that you actually can do with systems of this sort everything that you can do the other way, which is perhaps a little bit surprising. It looks as if the other way you had a lot more freedom. You just had to figure out more devious ways of getting away with things and then all was available to you.

But let me give you a feeling for what unification turns out to be like, with what we call 'referentially transparent' variables—referentially transparent because if somebody said "What is the value of this variable?" you don't have to say, "Well, when? What are you talking about? What is it now? What was it yesterday?" No, what is it? Always the same, according to this theory.
It's like this. Suppose that one of your intelligence operatives in a far-off country sent back some photographs of something that you needed information about, but the photographs had been damaged in transport, so that some of the emulsion had come off them so that you can see it all except for one area which has all been sort of scraped off and it is just white and the light passes through it and there is no detail there. Fortunately, you have another slide of exactly the same scene, taken from exactly the same angle. And it has been damaged, too, unfortunately, but the damage is in a different place, in this case. What would you do? Well, what you would do is pretty clear in that you would take one photograph and would superimpose it on the other and if you got the registration just right both of the damaged areas would disappear and you would see the whole thing.

That is what unification is like. Since you can never change the value of a variable, all you can ever do is to take an existing picture, and add more detail to it. There are things that you don't know about it, you can add things to it so that you know more about it, but you can never take something and change it. It has always got to stay. It is only the pieces that you do not know anything about that can be filled in.

So, you can take a string of words and in the course of analyzing it, you can fill in things that you don't know such as that a given unit is a piece of structure, or that another is a phrase, and so on. You can discover what the labels on those phrases are. Now, notice that context-free grammar—that is the grammar that contains only rules like the famous

\[ S \rightarrow NP \ VP \]

rule—that does not violate any of this. That is the most simple kind of unification grammar there is. You can check out whether one of these rules is correctly applied to a particular structure by simply looking at the piece of structure that that rule would have sanctioned. You don't have to say, well, we have to work down from the top, or we have to work up from the bottom, or we have to go left to right; you do not have to do anything. You can just pick some piece of that tree and you can say, "Well, wait a minute, what rule is responsible for this?" And you go look at the rule and if the rule is there, then that's all right and if it's not, it's not. You don't have to go through a certain set sequence of procedures.

The question that arises then is can we do all the things that the people with these more powerful techniques, so-called non-monotonic techniques—never mind about that word, that's the bad guys; the non-monotonic guys are the bad guys, and I'm talking about the monotonic guys and they are the good guys. The question that arises is "Can the good guys do all the things that the bad guys were trying to do?"—the advantages of being able to do it would be tremendous. That would, for example, mean that the theory of language that you wound up with would be the same theory for the speaker of the language as for the understander of the language. And I don't know about you but that sounds to me like rather a good idea. Because without it, you see, in principle, you could have somebody who had learned how to speak English and understand French and could not do the
other thing with either of the languages. It would be possible to 
communicate with them but it would be sort of funny and roundabout, 
wouldn't it? But it would be perfectly possible. You would have to 
account for the fact, if you don't have a theory like this, that if you 
teach a certain grammatical construction to somebody, if you teach him how 
to say this, if you teach him how to use this word, or you teach him how to 
use this tense, or you teach him how to use this construction, and he 
learns how to use it with great fluency and facility, you would then have 
to say to him, "Okay, now we can go on to understanding that. Let me teach 
you how to understand it." Now most people would say, "That's funny. 
You've just taught me that. I can understand it." It seems that somehow 
our theories of languages ought to be like that. Monotonic theories of 
language are like that quite naturally.

So let me write some rules for you. We're going to change gears now 
and get down to some specifics and I don't know how far I will get with 
this but I hope that some of these rules will just give you a feel for what 
you can do. And the rules are going to look like sort of decorated complex 
tree rules. There is going to be a rule in the simplest possible 
context-free grammar that you can imagine that says that a verb phrase can 
consist of a verb and a noun phrase:

\[ VP \rightarrow V \ NP \]

Yes, there are intransitive verbs and there will be other rules for verb 
phrases and subject-aux inversion and there are lots of other fancy things, 
but there will be this rule amongst all the rest. Now I am going to write 
another more fancy version of this rule by replacing each of the three 
things in it by something else. And at first it won't look as though I am 
doing very much to these rules except using up more chalk on them.

So, for example, what is a verb phrase? Well, it is something whose 
category is verb phrase. That, at least, we know about it. And in due 
course I am going to fill in more of the photograph. I am going to give 
you more of the emulsion down here. But I am not going to do it just yet. 
So, that is going to be one description. And the next thing in the rule is 
going to tell you about something called the category verb, and that will 
have some information to be added to it. The next thing is something whose 
category is noun phrase. All right, that much is simply a translation of 
what we have already.

\[
\begin{cases}
\text{CAT} = \text{VP} \\
\text{CAT} = V \\
\text{CAT} = \text{NP}
\end{cases}
\]

Now, different verbs take, let us say, different things. Some of them 
take a dative object and some of them require certain kinds of complements 
like 'John is easy to please'--you have probably heard that one. They take 
to-infinitives. They take other kinds of infinitives. Some of them take 
certain kinds of prepositions--you know all that sort of stuff. What we 
would like to have to do is to avoid writing a separate decorated rule for 
all the different things that verbs might take. We want this object NP to 
be any of the NPs that it might take next in line. If the verb wants a
dative NP then we would like it to be dative. We would like a lot of things of that sort to fall out. So I am going to assume that there is a thing called a 'frame'--and this is a Fillmorian-like notion--and the frame that this particular verb has, I am going to mention two parts of it: I'm going to mention 's', I'm going to write a dot, and I'm going to write the word 'rest'. And I'm going to put a question mark after each of these things.

\[
\begin{align*}
\text{CAT} & = \text{VP} \\
\text{FRAME} & = (s? \cdot \text{rest}) \\
\text{HF} & = h? = [o? = \text{obj}?] \\
\end{align*}
\]

Now let me explain all that for you. First of all, question mark is simply my way of bringing home to you that it is supposed to be a variable, something that can have any value. Next, this dot is used in the following way: the things in parentheses are lists of things. So whatever is the first thing in this list has got to be the value of the variable 's'. I don't know what it is, but the variable 's' is going to have that value. ('s' stands for subject and 'o' for object.) And the dot means that the next thing that follows here is not the next thing in the list; it is the rest of the list. And I have called that 'rest'. The thing is also going to have an attribute--these things to the left of the equals sign I call 'attributes'--called 'HF' and that stands for 'head features'. And that's for historical reasons. And they are going to do the same as whatever the variable 'h' has, and in particular, there is going to be an attribute and a value pair which is going to be at least part of what that set of head features consists of--and this is the worst piece of complexity; you are about to see the worst of the whole thing--that is going to say 'o? = obj?'.

Now each of these attributes is part of the picture. There is no limit to how complete we can make this picture. We can always add new attributes to any set that already exists. But if we add a new attribute to the picture, we must make sure that its value is consistent with the value we have already given it. We can't have an attribute 'A' with value 'a' and the same attribute 'A' in the same list with the value 'b', because that would be inconsistent. But the head features here could have itself another attribute other than the one that is the value of this variable 'o'. And so long as it is a different attribute it can have whatever value it likes.

Okay, now how are we going to make sense out of all this? Well, the verb itself also has a frame, which will consist of 's?', 'o?', and the 'rest'. And it is going to have some head features (HF) and it is going to have the value 'h':

\[
\begin{align*}
\text{CAT} & = \text{V} \\
\text{FRAME} & = (s? \ o? \cdot \text{rest}) \\
\text{HF} & = h? \\
\end{align*}
\]
Let me rush through the rest of this now. And this NP (following V) is just going to have some head features. And it is going to have the value 'obj'.

\[
\begin{align*}
\text{CAT} &= \text{NP} \\
\text{HF} &= \text{obj}?
\end{align*}
\]

This means an awful lot. This is going to just match the object here. Whatever object is picked for this sentence here is going to have to--and now let me really use the word--it is going to have to 'unify' with the variable 'obj'. That means it can have whatever attributes and values it likes just so long as it has this (NP) value for the category feature. I don't care about the rest. But for the category feature it has this NP value, and for the head feature attribute it must have...anything it likes, so long as it is unifiable with the value of the variable 'obj'.

\[
\begin{align*}
\text{CAT} &= \text{VP} \\
\text{FRAME} &= (s? \ . \ rest?) \\
\text{HF} &= h? = [o? = \text{obj}?] \rightarrow \\
\text{CAT} &= \text{V} \\
\text{FRAME} &= (s? \ o? \ . \ rest?) \\
\text{HF} &= h? \\
\text{CAT} &= \text{NP} \\
\text{HF} &= \text{obj}?
\end{align*}
\]

Now so far we don't know anything about the variable object and maybe this is all we will ever learn about it. So, just for the time being, let us assume that whatever we find out about this noun phrase, it has these interesting features that we care about and they are simply going to become the value of this variable. The only things that can happen to destroy that idea is if we discover later some further details about the value of this variable. Remember it can never change, we can just learn more about it. That's all we can ever do.

Well, it turns out that that value 'obj' crops up in one other place, namely in the head features of the verb phrase. So, whatever we learn about that noun phrase here--and I am going to assume that the values of head features are the really meaty stuff about it that we really care about--is going to become part of the head features of this noun phrase that we are interested in building; in particular, it is going to become the value of some attribute; unfortunately we don't yet know which attribute. Well, for that we look at the verb frame. Suppose that the frame of the particular verb that we were looking at said that it has an agent and it has a patient, so that it is a standard transitive verb. Well, what that would say is, you would take the second thing in the verb frame--that is the 'o' thing--and that 'o' has to unify with the attribute over here in the VP HF. So that means that this noun phrase is going to become the patient. Notice that it is the verb that decides what it is going to become. Suppose that it was required that it should be in the accusative case. Yet all we would have to do here is to say that the verb has a restriction on it to the effect that the patient must be case=accusative:
And that would mean that the only thing that we would be able to unify with the patient attribute here was something which has this further attribute. So long as that unification could go through, then it would be all right to take that. Notice that we don't have to do anything; we can say anything here we like, about any of the things that the verb picks. You can say they have to be animate or they have to be human or they have to be speckled pink or whatever it happens to be. So that now we know that the head features of the verb phrase have to unify with whatever the head features of the verb happen to be. And in that way this is a feature; in fact, that is why they call it head features, because the verb we regard as being the head of the verb phrase, and the verb phrase in turn would become the head of the sentence. And the head features would be copied over in that case also.

Now let me show you just one other small thing that you buy with this—I've got lots of other examples here that are so meaty and interesting and you won't get to see any of them—how does passive work in a system like this? Remember we have all of these verbs all of which can say whatever they like about the various complements that they take. Well, we regard an English verb as having two kinds of past participles that generally look the same, just the one which is used in perfect tenses and the one that is used for forming passives—let me not pursue that a great deal further for the moment. But there is a rule that tells you how to get from the active form of a verb to its passive form. It says that the verb has a certain frame, and I will say for the moment that that consists of the 'first' thing and the 'rest', so it can be anything, any list of things that contains at least one member, and it has some head features—well I won't spell out the details for the head features, they could be almost anything—that can be rewritten as, can be stated alternatively as, something which is a category verb, which is a frame which is 'rest' and head features which are the same as these (the active verb's) head features except that they have voice=passive:

\[
\begin{align*}
\text{CAT} &= \text{V} \\
\text{FRAME} &= (\text{first}? . \text{rest}?) \\
\text{HF} &= h? = [\text{voice} = \text{passive}] \\
\end{align*}
\]

Now passive in English is done periphrastically; that is to say it is done by putting certain auxiliary verbs in front of it and it does not get to be passive until that happens. But when you put the passive auxiliary verb in front of it they are also going to say that it has to be voice=passive. And so it is only with this form of the verb that the unification will go through. So we don't have to worry about that aspect of it. The key thing that we have done here is to take the verb in its original active form which consisted of some number of things, the 'first' and the 'rest', and we have simply replaced it by the 'rest'. All right, we have removed the first thing from that list. What that means is that by this rule here
(the unified VP rule above), the next thing, if there were going to be a complement, it would have to be not the second thing anymore, but the third thing, because the thing that used to be the second thing is now the first thing, and it will be the subject. So what used to be the object will now automatically become the subject. This sort of maneuver works for dative movement and translations such as NP complement into VP complements and other technical things of that sort.

I now consider myself to have one minute in which to show you very briefly of unification working in a totally different environment, namely morphology. Consider that you cannot normally reverse rewriting rules. So a rule that says that a y in certain circumstances on the end of an English word was rewritten as ie, for example, in the environment preceding a vowel or the funny S previously spoken about:

\[
\begin{align*}
y & \rightarrow \text{ie} / \left\{ \text{Vowel} \right\} \\
& \left\{ S \right\}
\end{align*}
\]

This rule might occur in an ordered list of rules of this sort; in particular, the funny S rule has got to follow it, hasn't it, because it is still there when this rule applies. These rules are not generally reversible. Let me give you a monotonic version of this same rule. What I will give you is a state diagram, which says you can rewrite a y--actually this is going to be part of this rule; it would take another one to finish it--you can rewrite an i into a y, provided that immediately following it you rewrite nothing at all as an e:

\[
\begin{align*}
s & \rightarrow \text{S} \\
a & \rightarrow \text{a} \\
\end{align*}
\]

So you combine to use both the i and the e, provided that what follows this brings you back to the beginning, either rewrites the funny S as a funny S, or takes some vowel and produces that same vowel as output. If, on the other hand, you were to find on the input a y and were to produce a y on the output so this would be a y that was going to be dealt with not by
this rule but by some subsequent rule, then that's all right, provided that what follows is in fact not the funny S or a vowel, because if you failed to do the correct translation in the case where what follows was one of these things, then you would have done something wrong; that is the case where the rule ought to have applied and you wind up in a state from which there is no way out. If you find anything else, you come back to the beginning. All I have done is to translate that rule into what is called a 'finite-state transducer'. So that now we can take the input to the rewriting rules from which we intend to get some output. And we can put it through a transducer equivalent to the first rule and another one equivalent to the second rule and as we have seen the step here is part of that rule—you haven't actually seen that but I'll tell you it is—you can draw another diagram right here and you keep on like this until you have dealt with all the rules:

![Diagram](image)

Now this is a mathematical object of really rather different character than the rewriting rules, and thanks to that different character there is an algorithm that we can perform on two adjacent members of this sequence here and make one out of them. It means that we no longer have to bother about what would have existed in between them. And we can keep on doing that until there is only one left. We can get one transducer of exactly this form, no more complicated in its general structure, but of course very much larger, and we can be completely indifferent as to whether it takes this string into that string or that string into this string. It is, in other words, a completely monotonic device, and it comes about because this is a unification statement of that rewriting statement (y --> \(\ldots\))—notice nothing ever changes in this. One merely chooses the right set of statements to go to. I have now overshot my time by 58 seconds and I invite questions fast.