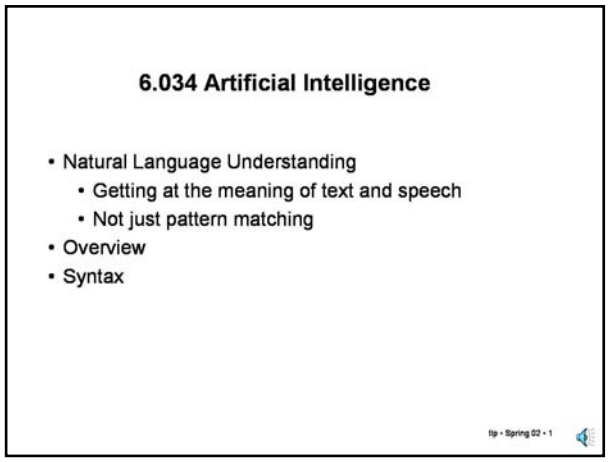
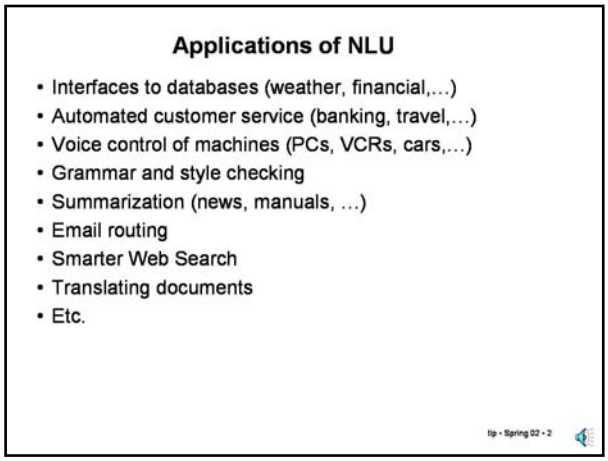
6.034 Notes: Section 12.1

Slide 12.1.1



In this chapter, we take a quick survey of some aspects of natural language understanding. Our goal will be to capture the **meaning** of sentences in some detail. This will involve finding representations for the sentences that can be connected to more general knowledge about the world. This is in contrast to approaches to dealing with language that simply try to match textual patterns, for example, web search engines.

We will briefly provide an overview of the various levels and stages of natural language processing and then begin a more in-depth exploration of language syntax.

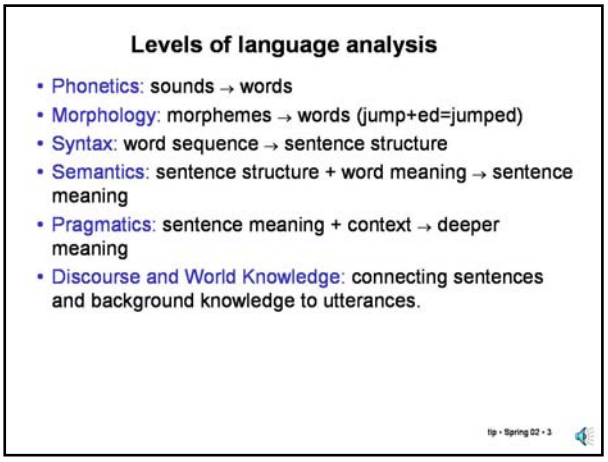


Slide 12.1.2

The motivation for the study of natural language understanding is twofold. One is, of course, that language understanding is one of the quintessentially human abilities and an understanding of human language is one of key steps in the understanding of human intelligence.

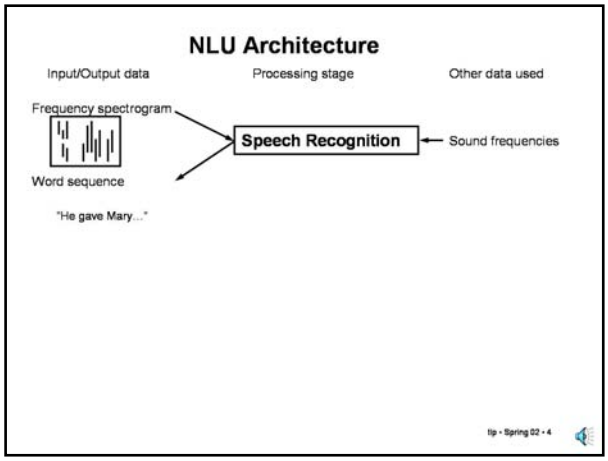
In addition to this fundamental long-term scientific goal, there is a pragmatic shorter-term engineering goal. The potential applications of in-depth natural language understanding by computers are endless. Many of the applications listed here are already available in some limited forms and there is a great deal of research aimed at extending these capabilities.

Slide 12.1.3



Language is an enormously complex process, which has been studied in great detail for a long time. The study of language is usually partitioned into a set of separate sub-disciplines, each with a different focus. For example, phonetics concerns the rules by which sounds (phonemes) combine to produce words. Morphology studies the structure of words: how tense, number, etc is captured in the form of the word. Syntax studies how words are combined to produce sentences. Semantics studies how the meaning of words are combined with the structure of a sentence to produce a meaning for the sentence, usually a meaning independent of context. Pragmatics concerns how context factors into the meaning (e.g. "it's cold in here") and finally there's the study of how background knowledge is used to actually understand the meaning the utterances.

We will consider the process of understanding language as one of progressing through various "stages" or processing that break up along the lines of these various subfields. In practice, the processing may not be separated as cleanly as that, but the division into stages allows us to focus on one type of problem at a time.



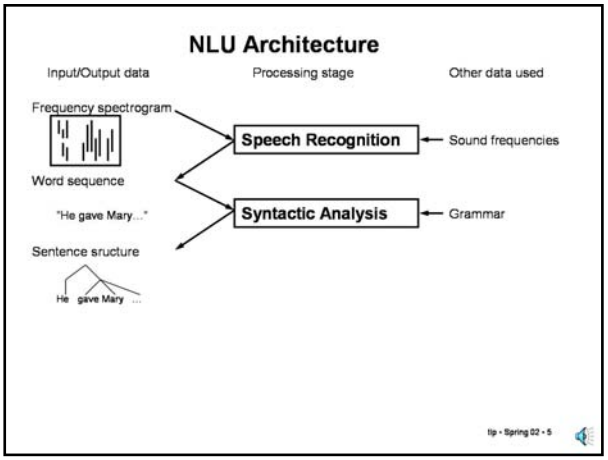
Slide 12.1.4

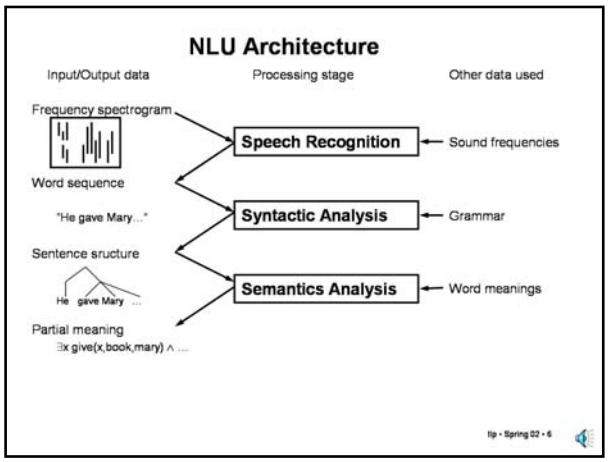
If one considers the problem of understanding speech, the first stage of processing is, conceptually, that of converting the spoken utterance into a string of words. This process is extremely complex and quite error prone and, today, cannot be solved without a great deal of knowledge about what the words are likely to be. But, in limited domains, fairly reliable transcription is possible. Even more reliability can be achieved if we think of this stage as producing a few alternative interpretations of the speech signal, one of which is very likely to be the correct interpretation.

Slide 12.1.5

The next step is **syntax**, that is, computing the structure of the sentence, usually in terms of phrases, such as noun phrases, verb phrases and prepositional phrases. These nested phrases will be the basis of all subsequent processing. Syntactic analysis is probably the best developed area in computational linguistics but, nevertheless, there is no universally reliable "grammar of English" that one can use to parse sentences as well as trained people can. There are, however, a number of wide-coverage grammars available.

We will see later that, in general, there will not be a unique syntactic structure that can be derived from a sequence of words.



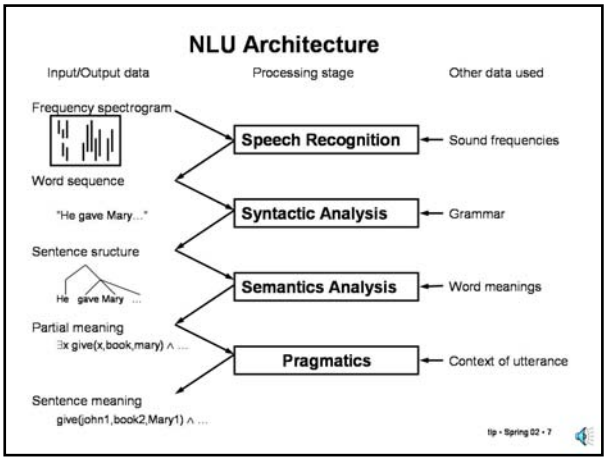


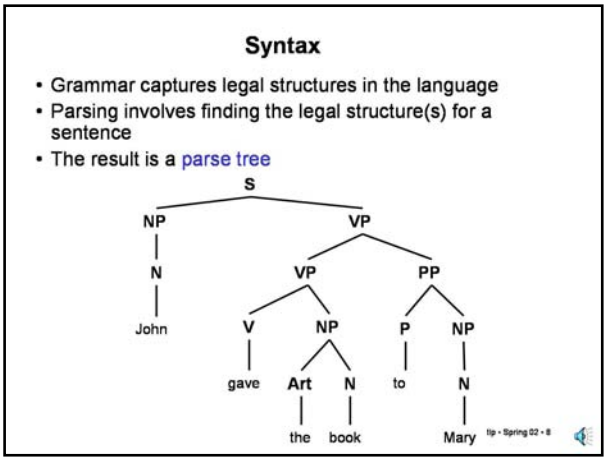
Slide 12.1.6

Given the sentence structure, we can begin trying to attach meaning to the sentence. The first such phase is known as **semantics**. The usual intent here is to translate the syntactic structure into some form of logical representation of the meaning - but without the benefit of context. For example, who is being referred to by a pronoun may not be determined at this point.

Slide 12.1.7

We will focus in this chapter on syntax and semantics, but clearly there is a great deal more work to be done before a sentence could be understood. One such step, sometimes known as **pragmatics**, involves among other things disambiguating the various possible senses of words, possible syntactic structures, etc. Also, trying to identify the referent of pronouns and descriptive phrases. Ultimately, we have to connect the meaning of the sentence with general knowledge in order to be able to act on it. This is by far the least developed aspect of the whole enterprise. In practice, this phase tends to be very application specific.





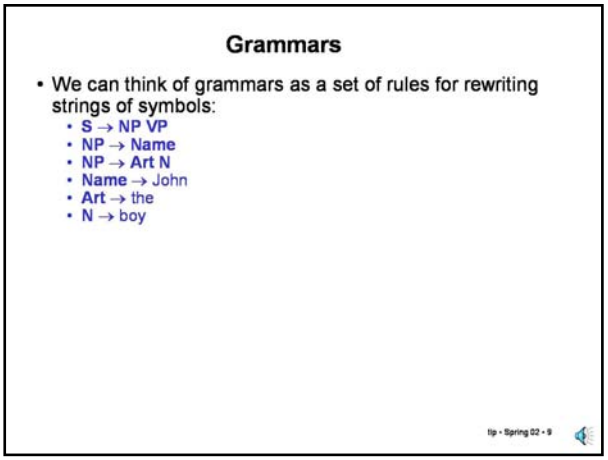
Slide 12.1.8

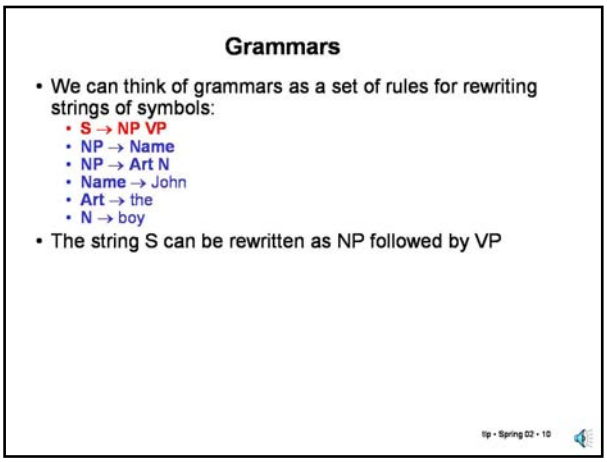
In the rest of this section, we will focus on syntax. The description of the legal structures in a language is called a **grammar**. We'll see examples of these later. Given a sentence, we use the grammar to find the legal structures for a sentence. This process is called **parsing** the sentence. The result is one or more **parse trees**, such as the one shown here, which indicates that the sentence can be broken down into two **constituents**, a noun phrase and a verb phrase. The verb phrase, in turn, is composed of another verb phrase followed by a prepositional phrase, etc.

Our attempt to understand sentences will be based on assigning meaning to the individual constituents and then combining them to construct the meaning of the sentence. So, in this sense, the constituent phrases are the atoms of meaning.

Slide 12.1.9

A grammar is typically written as a set of **rewrite rules** such as the ones shown here in blue. Bold-face symbols, such as S, NP and VP, are known as non-terminal symbols, in that they can be further re-written. The non-bold-face symbols, such as John, the and boy, are the words of the language - also known as the terminal symbols.



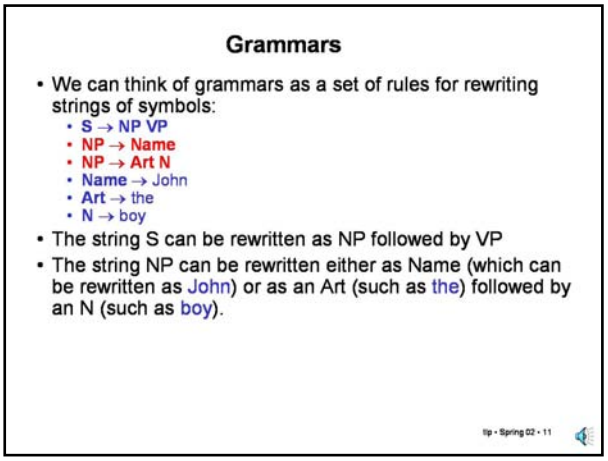


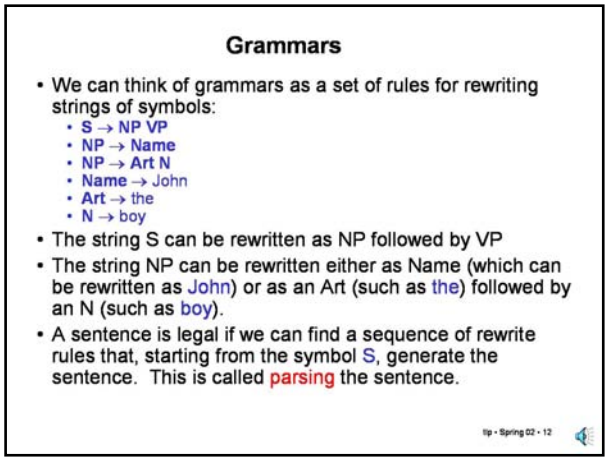
Slide 12.1.10

The first rule, S -> NP VP, indicates that the symbol S (standing for sentence) can be rewritten as NP (standing for noun phrase) followed by VP (standing for verb phrase).

Slide 12.1.11

The symbol NP, can be rewritten either as a Name or as an Art(icle), such as the, followed by a N (oun), such as boy.





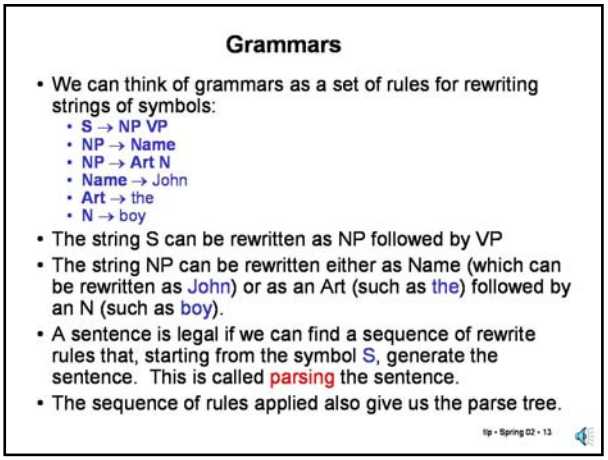
Slide 12.1.12

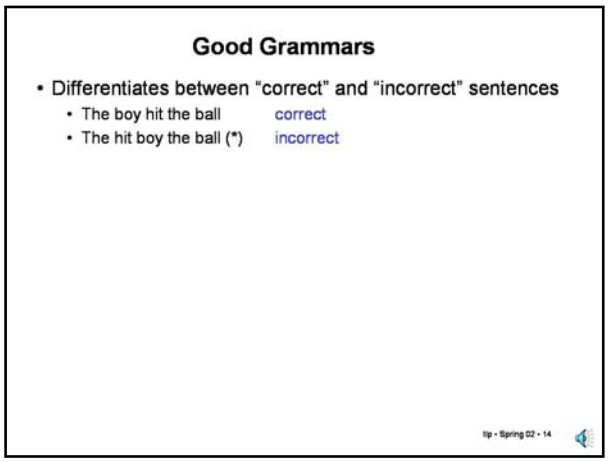
If we can find a sequence of rewrite rules that will rewrite the initial S into the input sentence, the we have successfully parsed the sentence and it is legal.

Note that this is a search process like the ones we have studied before. We have an initial state, S, at any point in time, we have to decide which grammar rule to apply (there will generally be multiple choices) and the result of the application is some sequence of symbols and words. We end the search when the words in the sentence have been obtained or when we have no more rules to try.

Slide 12.1.13

Note that the successful sequence of rules applied to achieve the rewriting give us the parse tree. Note that this excludes any "wrong turns" we might have taken during the search.





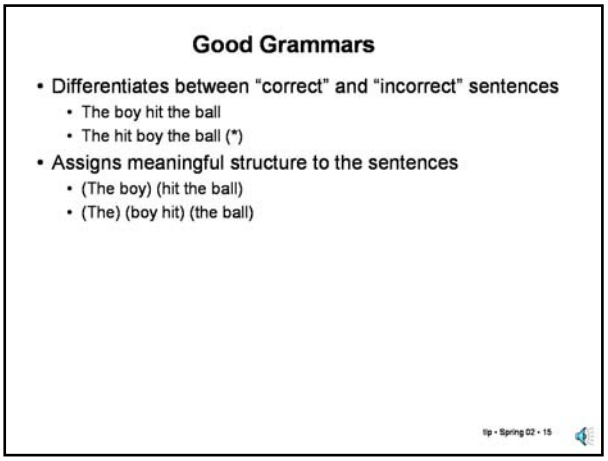
Slide 12.1.14

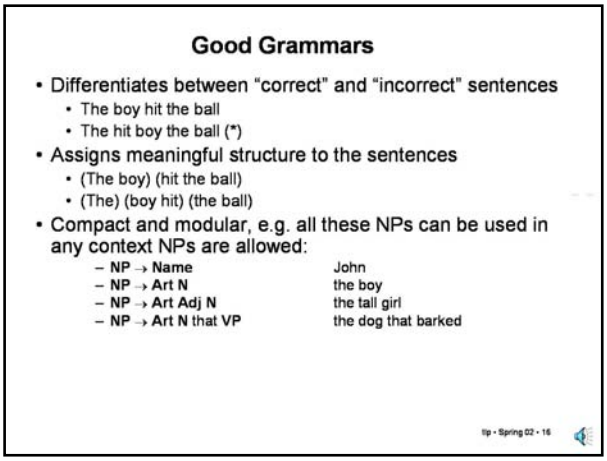
What makes a good grammar?

The primary criterion is that it differentiates correct sentences from incorrect ones. (By convention an asterisk next to a sentence indicates that it is not grammatical).

Slide 12.1.15

The other principal criterion is that it assigns "meaningful" structures to sentences. In our case, this literally means that it should be possible to assign meaning to the sub-structures. For example, a noun phrase will denote an object while a verb phrase will denote an event or an action, etc.





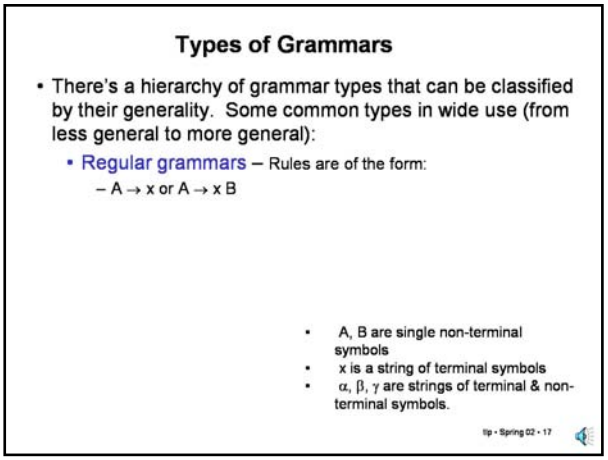
Slide 12.1.16

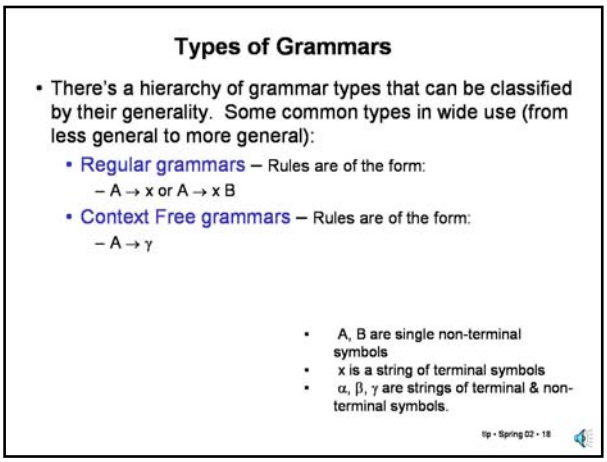
Among the grammars that meet our principal criteria we prefer grammars that are compact, that is, have fewer rules and are modular, that is, define structures that can be re-used in different contexts - such as noun-phrase in this example. This is partly for efficiency reasons in parsing, but is partly because of Occam's Razor - the simplest interpretation is best.

Slide 12.1.17

There are many possible types of grammars. The three types that are most common in computational linguistics are regular grammars, context-free grammars and context-sensitive grammars. These grammars can be arranged in a hierarchy (the Chomsky hierarchy) according to their generality. In this hierarchy, the grammars in higher levels fully contain those below and there are languages in the more general grammars not expressible in the less general grammars.

The least general grammar of some interest in computational linguistics are the **regular grammars**. These grammars are composed of rewrite rules of the form A -> x or A -> x B. That is, a non-terminal symbol can be rewritten as a string of terminal symbols or by a string of terminal symbols followed by a non-terminal symbol.





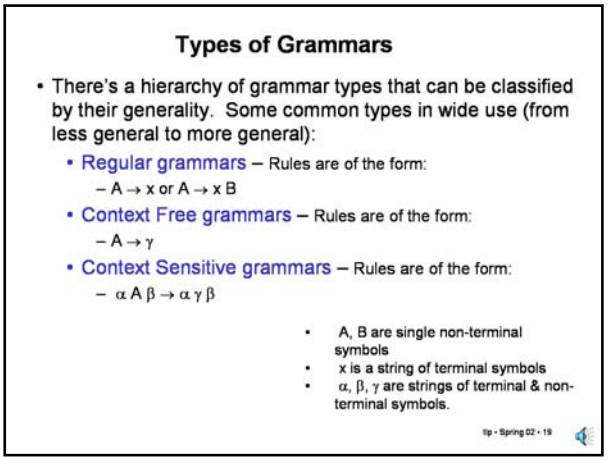
Slide 12.1.18

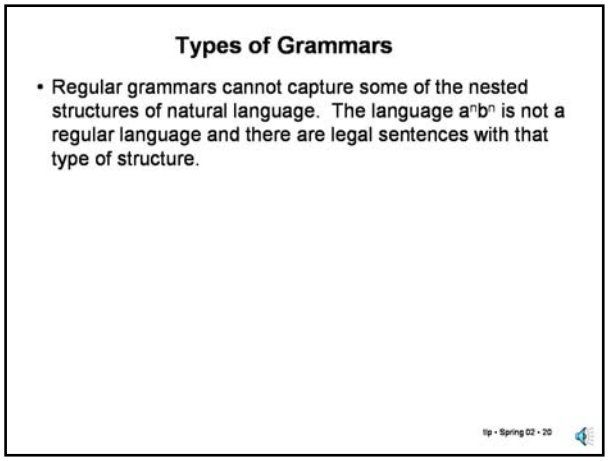
At the next level are the **context-free grammars**. In these grammars, a non-terminal symbol can be rewritten into any combination of terminal and non-terminal symbols. Note that since the non-terminal appears alone in the left-hand side (lhs) of the rule, it is re-written independent of the context in which it appears - and thus the name.

Slide 12.1.19

Finally, in **context-sensitive** grammars, we are allowed to specify a context for the rewriting operation.

There are even more general grammars (known as Type 0) which we will not deal with at all.



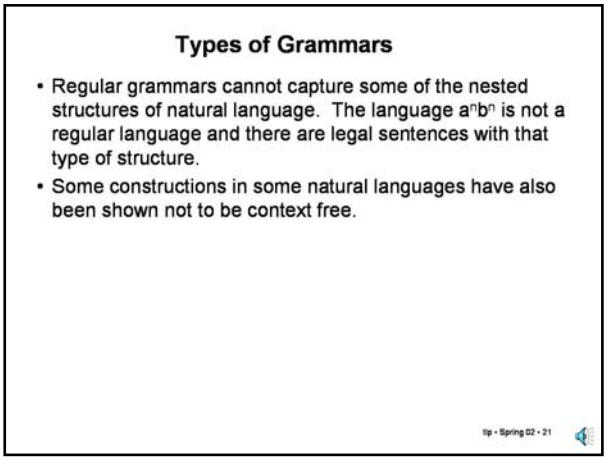


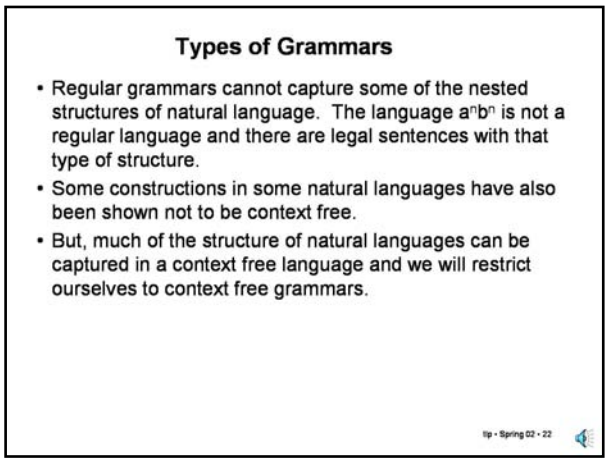
Slide 12.1.20

The language of parenthesized expressions, that is, n left parens followed by n right parens is the classic example of a non-regular language that requires us to move to context-free grammars. There are legal sentences in natural languages whose structure is isomorphic to that of parenthesized expressions (the cat likes tuna; the cat the dog chased likes tuna; the cat the dog the rat bit chased likes tuna). Therefore, we need at least a context-free grammar to capture the structure of natural languages.

Slide 12.1.21

There have been several empirical proofs that there exist natural languages that have non-context-free structure.



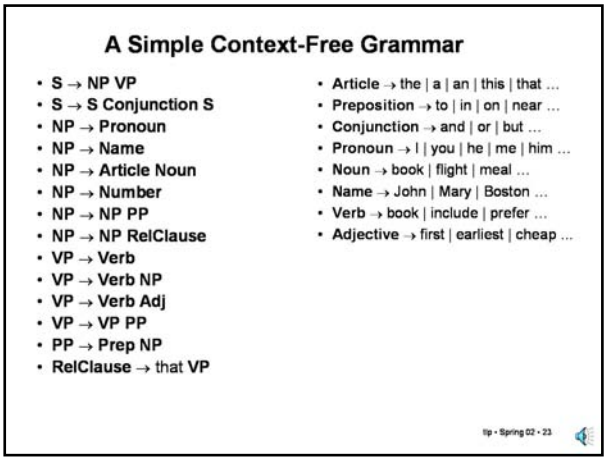


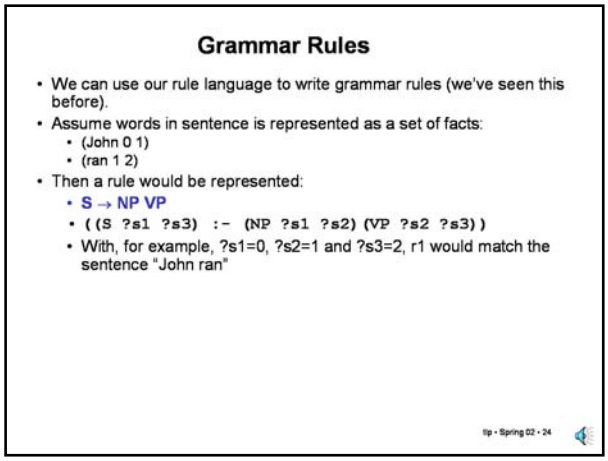
Slide 12.1.22

However, much of natural language can be expressed in context-free grammars extended in various ways. We will limit ourselves to this class.

Slide 12.1.23

Here's an example of a context free grammar for a small subset of English. Note that the vertical band is a short hand which can be read as "or"; it is a notation for combining multiple rules with identical left hand sides. Many variations on this grammar are possible but this illustrates the style of grammar that we will be considering.





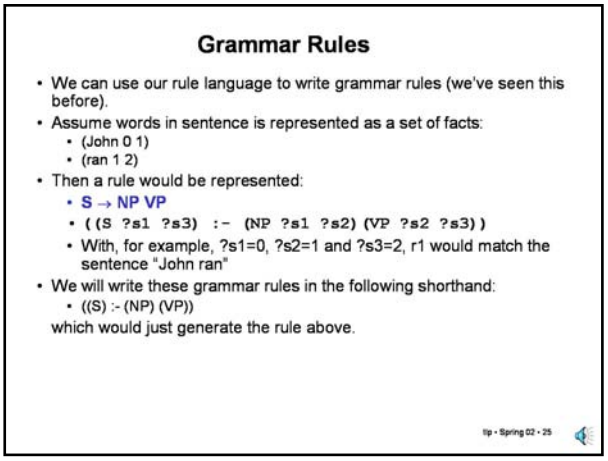
Slide 12.1.24

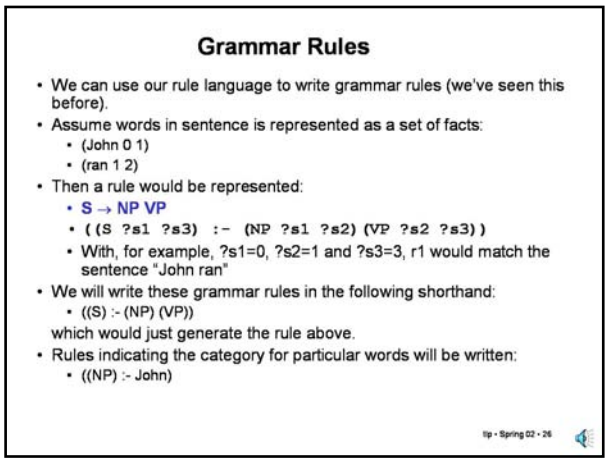
At this point, we should point out that there is a strong connection between these grammar rules that we have been discussing and the logic programming rules that we have already studied. In particular, we can write context-free grammar rules in our simple Prolog-like rule language.

We will assume that a set of facts are available that indicate where the particular words in a sentence start and end (as shown here). Then, we can write a rule such as S -> NP VP as a similar Prolog-like rule, where each non-terminal is represented by a fact that indicates the type of the constituent and the start and end indices of the words.

Slide 12.1.25

In the rest of this Chapter, we will write the rules in a simpler shorthand that leaves out the word indices. However, we will understand that we can readily convert that notation into the rules that our rule-interpreters can deal with.



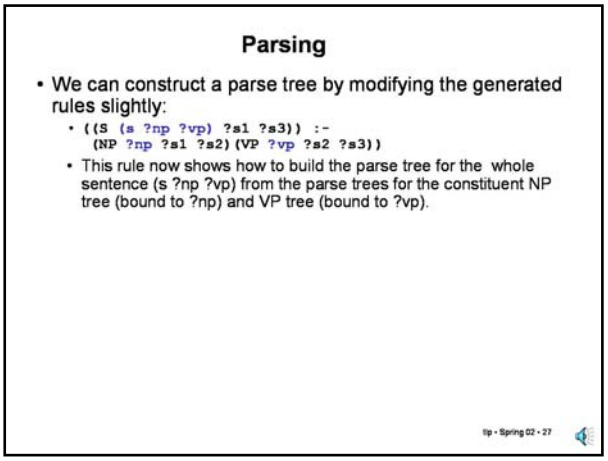


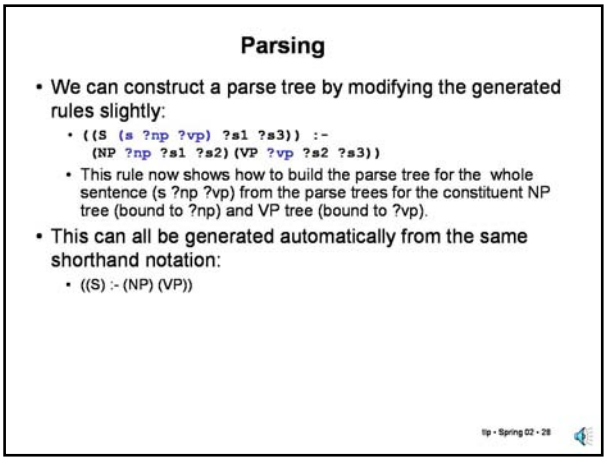
Slide 12.1.26

We can also use the same syntax to specify the word category of individual words and also turn these into rules.

Slide 12.1.27

We can make a small modification to the generated rule to keep track of the parse tree as the rules are being applied. The basic idea is to introduce a new argument into each of the facts which keeps track of the parse tree rooted at that component. So, the parse tree for the sentence is simply a list, starting with the symbol S, and whose other components are the trees rooted at the NP and VP constituents.



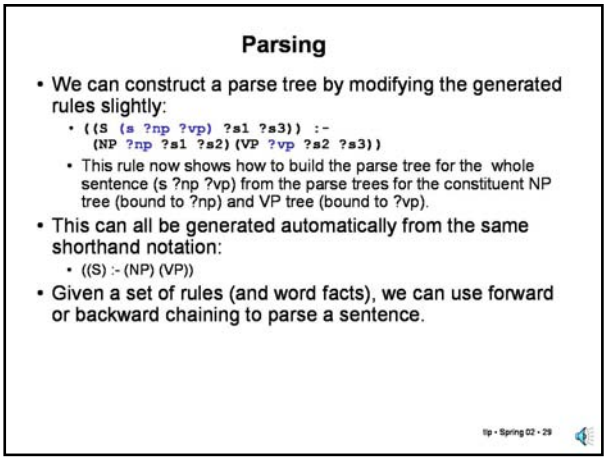


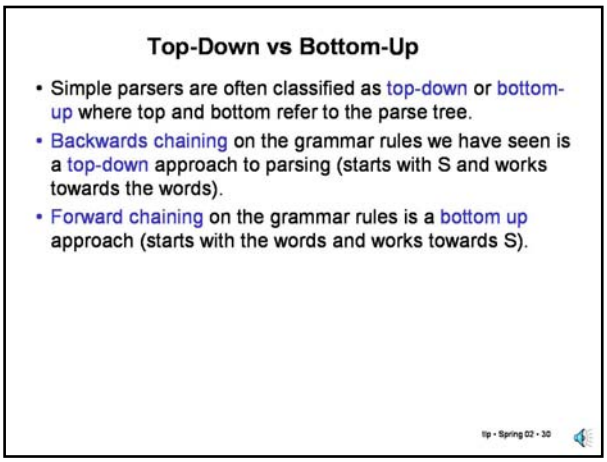
Slide 12.1.28

This additional bit of bookkeeping can also be generated automatically from the shorthand notation for the rule.

Slide 12.1.29

Note that given the logic rules from the grammar and the facts encoding a sentence, we can use chaining (either forward or backward) to parse the sentence. Let's look at this in more detail.



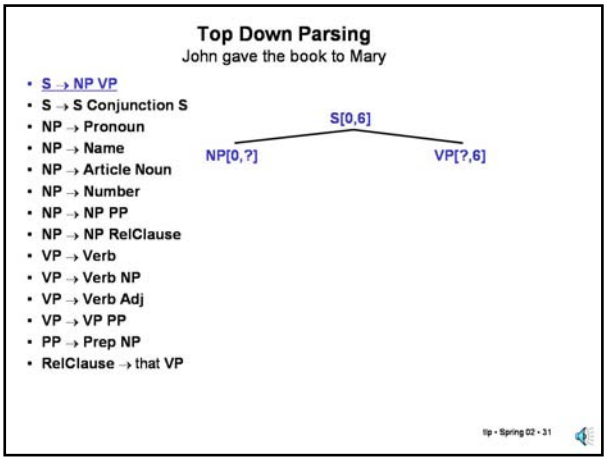


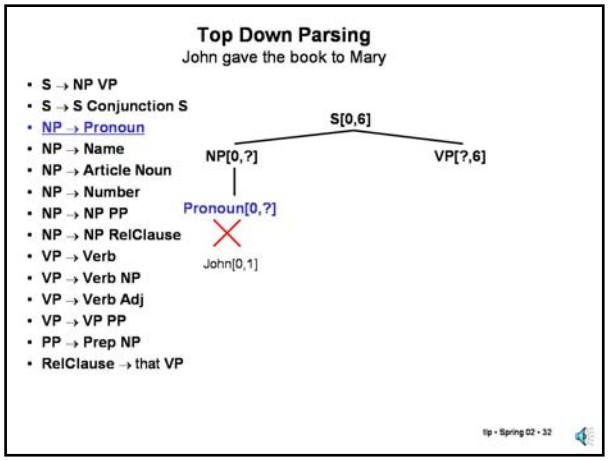
Slide 12.1.30

A word on terminology. Parsers are often classified into **top-down** and **bottom-up** depending whether they work from the top of the parse tree down towards the words or vice-versa. Therefore, backward-chaining on the rules leads to a top-down parser, while forward-chaining, which we will see later, leads to a bottom-up parser. There are more sophisticated parsers that are neither purely top-down nor bottom-up, but we will not pursue them here.

Slide 12.1.31

Let us look at how the sample grammar can be used in a top-down manner (backward-chaining) to parse the sentence "John gave the book to Mary". We start backchaining with the goal S[0,6]. The first relevant rule is the first one and so we generate two subgoals: NP[0,?] and VP[?,6].



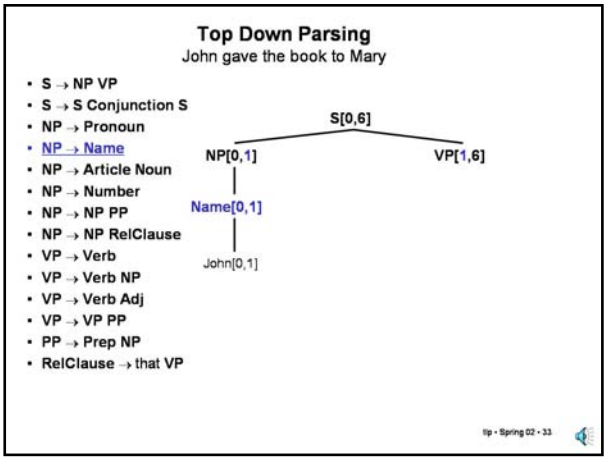


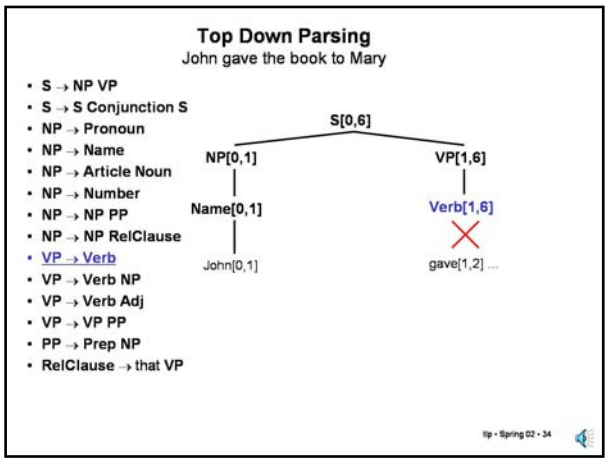
Slide 12.1.32

Assuming we examine the rules in order, we first attempt to apply the NP -> Pronoun rule. But that will fail when we actually try to find a pronoun at location 0.

Slide 12.1.33

Then we try to see if NP -> Name will work, which it does, since the first word is John and we have the rule that tells us that John is a Name. Note that this will also bind the end of the VP phrase and the start of the VP to be at position 1.



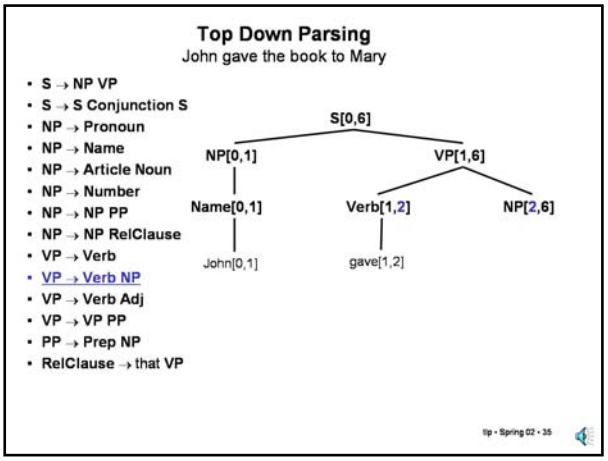


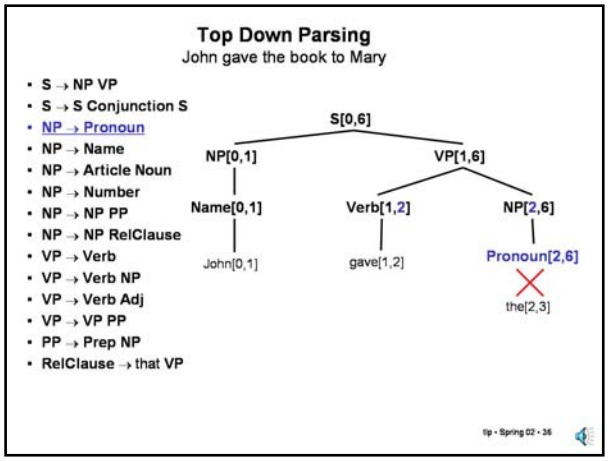
Slide 12.1.34

So, we move on to the pending VP. Our first relevant rule is VP -> Verb, which will fail. Note, however, that there is a verb starting at location 1, but at this point we are looking for a verb phrase from positions 1 to 6, while the verb only goes from 1 to 2.

Slide 12.1.35

So, we try the next VP rule, which will look for a verb followed by a noun phrase, spanning from words 1 to 6. The Verb succeeds when we find "gave" in the input.



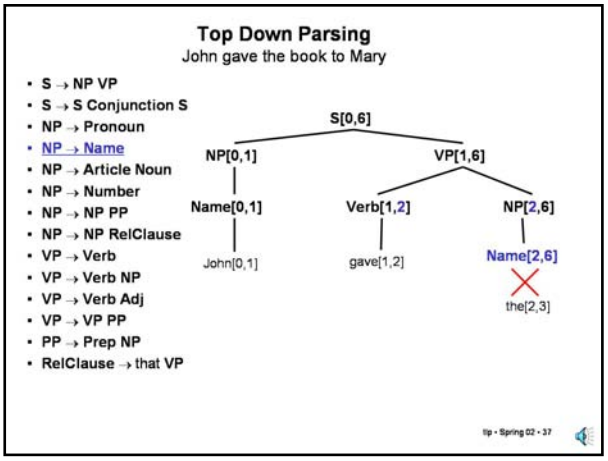


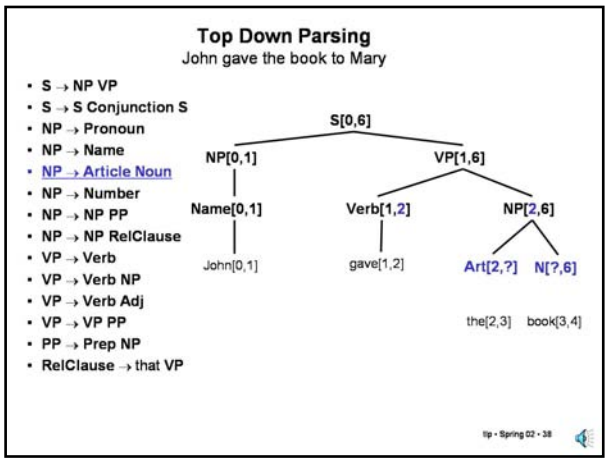
Slide 12.1.36

Now we try to find an NP starting at position 2. First we try the pronoun rule, which fails.

Slide 12.1.37

Then we try the name rule, which also fails.



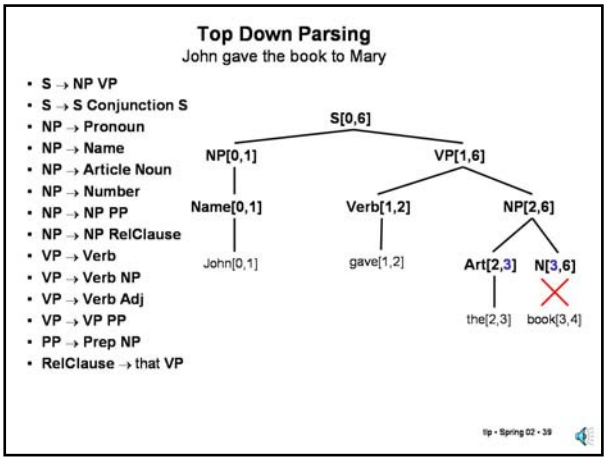


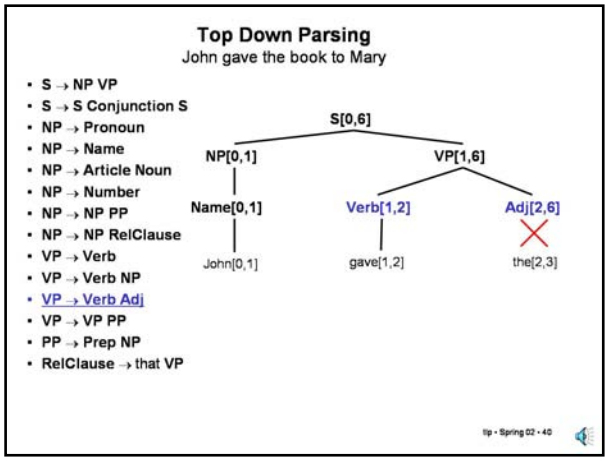
Slide 12.1.38

Then we try the article followed by a noun.

Slide 12.1.39

The article succeeds when we find "the" in the input. Now we try to find a noun spanning words 3 to 6. We have a noun in the input but it only spans one word, so we fail.



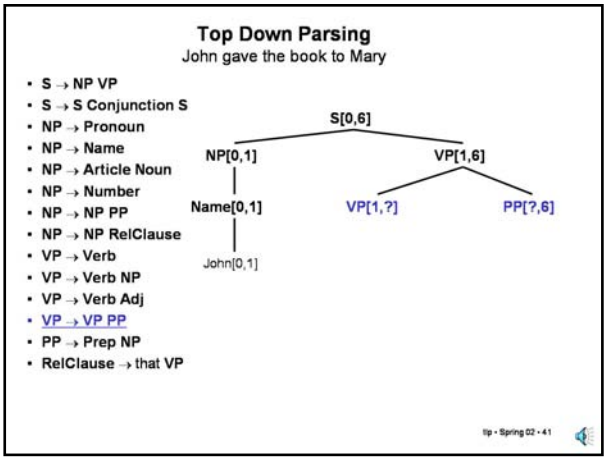


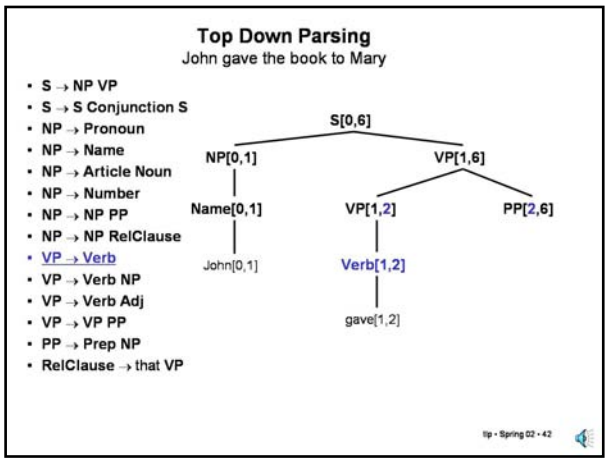
Slide 12.1.40

We eventually fail back to our choice of the VP rule and so we try the next VP rule candidate, involving a Verb followed by an adjective, which also fails.

Slide 12.1.41

The next VP rule, looks for a VP followed by prepositional phrase.



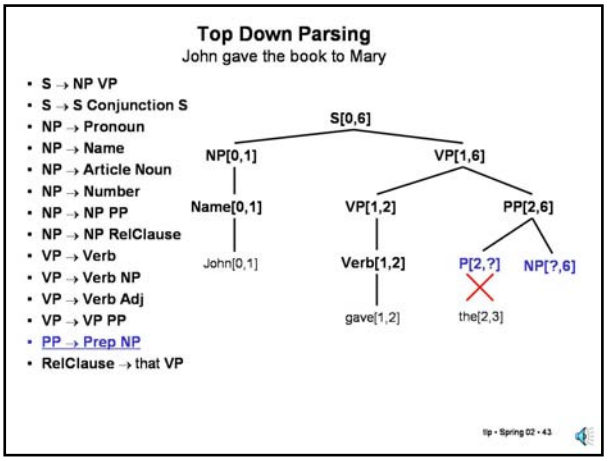


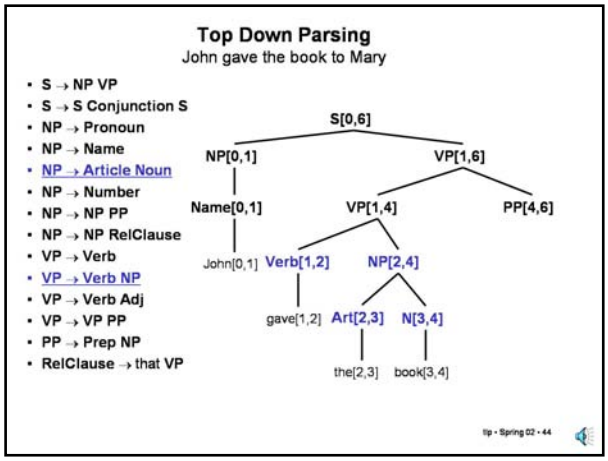
Slide 12.1.42

The first VP succeeds by finding the verb "gave", which now requires us to find a prepositional phrase starting at position 2.

Slide 12.1.43

We proceed to try to find a preposition at position 2 and fail.



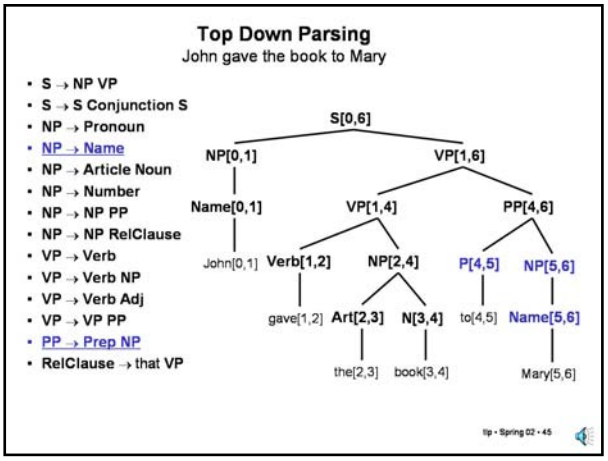


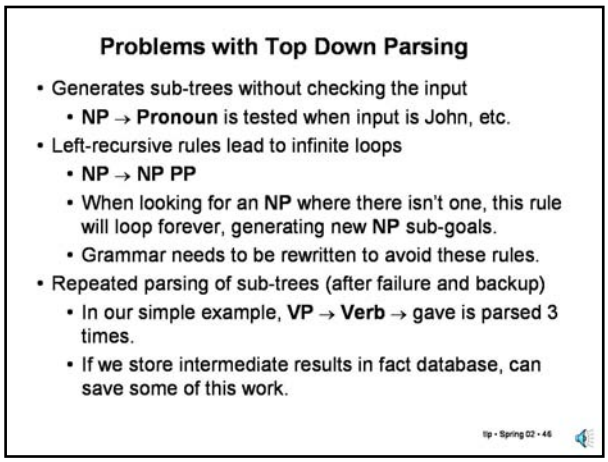
Slide 12.1.44

We fail back to trying an alternative rule (verb followed by NP) for the embedded VP, which now successfully parses "gave the book" and we proceed to look for a prepositional phrase in the range 4 to 6.

Slide 12.1.45

Which successfully parses, "to Mary", and the complete parse succeeds.





Slide 12.1.46

There are a number of problems with this top-down parsing strategy. One that substantially impacts efficiency is that rules are chosen without checking whether the next word in the input can possibly be compatible with that rule. There are simple extensions to the top-down strategy to overcome this difficulty (by keeping a table of constituent types and the lexical categories that can begin them).

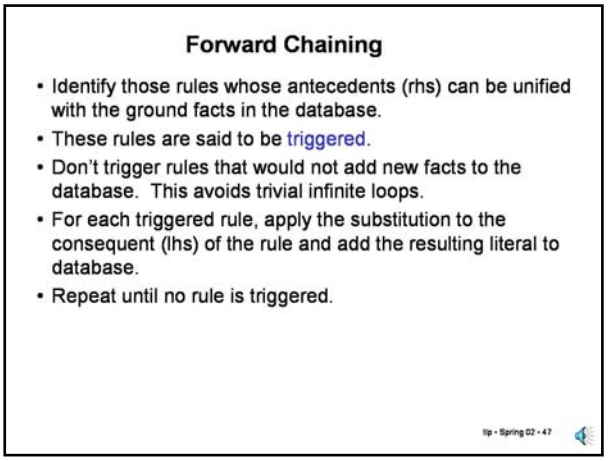
A more substantial problem, is that rules such as NP -> NP PP (left-branching rules) will cause an infinite loop for this simple top-down parsing strategy. It is possible to modify the grammar to turn such rules into right-branching rules - but that may not be the natural interpretation.

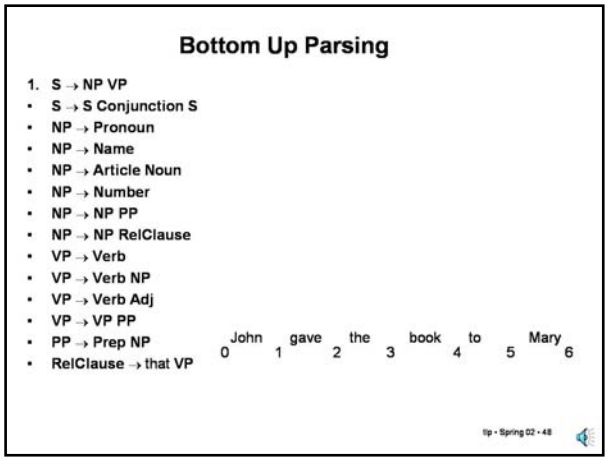
Note that the top-down strategy is carrying out a search for a correct parse and it ends up doing wasted work, repeatedly parsing parts of the sentence during its attempts. This can be avoided by building a table of parses that have been previously discovered (stored in the fact database) so they can be reused rather than re-discovered.

Slide 12.1.47

So far we have been using our rules together with our backchaining algorithm for logic programming to do top-down parsing. But, that's not the only way we can use the rules.

An alternative strategy starts by identifying any rules for which all the literals in their right hand side can be unified (with a single unifier) to the known facts. These rules are said to be **triggered**. For each of those triggered rules, we can add a new fact for the left hand side (with the appropriate variable substitution). Then, we repeat the process. This is known as **forward chaining** and corresponds to bottom-up parsing, as we will see next.



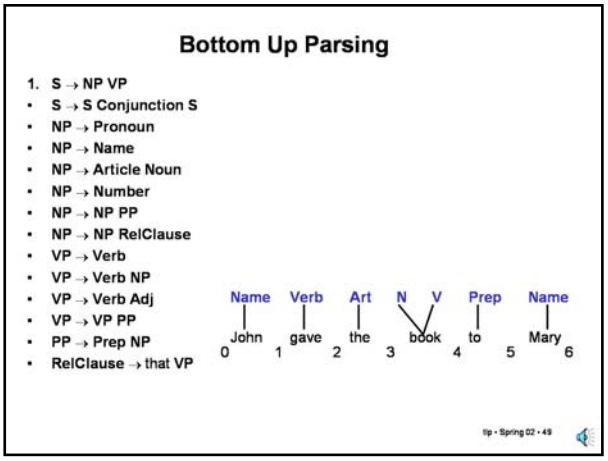


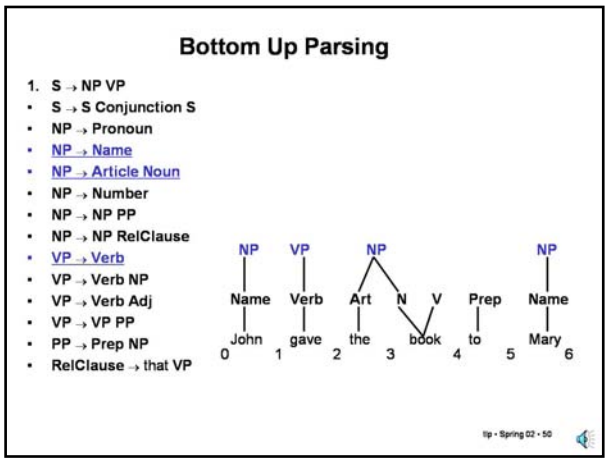
Slide 12.1.48

Now, let's look at bottom-up parsing. We start with the facts indicating the positions of the words in the input, shown here graphically.

Slide 12.1.49

Note that all the rules indicating the lexical categories of the individual words, such as Name, Verb, etc, all trigger and can all be run to add the new facts shown here. Note that book is ambiguous, both a noun and a verb, and both facts are added.



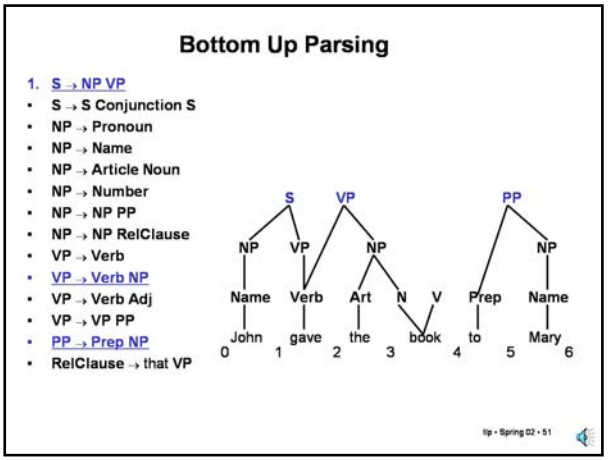


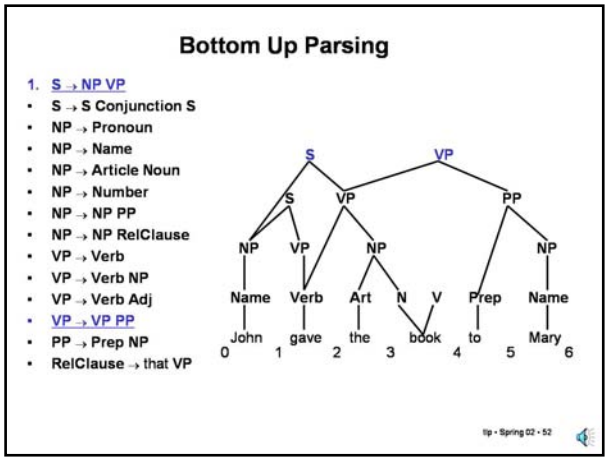
Slide 12.1.50

Now these three rules (NP -> Name, VP-> Verb and NP -> Art N) all trigger and can be run.

Slide 12.1.51

Then, another three rules (S -> NP VP, VP -> Verb NP and PP -> Prep NP) trigger and can be run. Note that we now have an S fact, but it does not span the whole input.



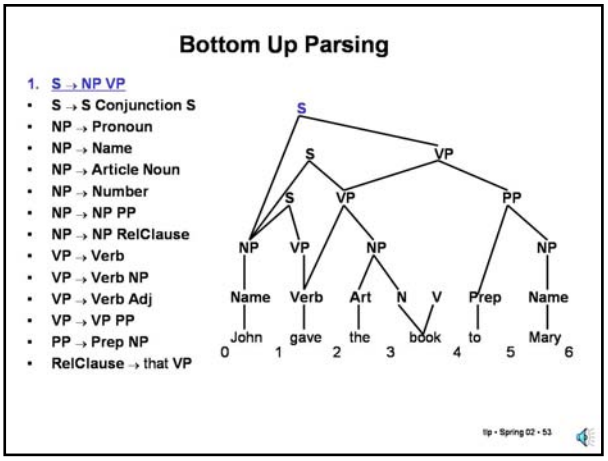


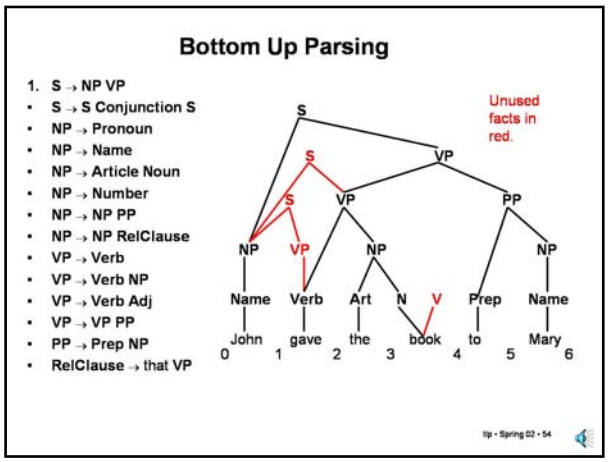
Slide 12.1.52

Now, we trigger and run the S rule again as well as the VP->VP PP rule.

Slide 12.1.53

Finally, we run the S rule covering the whole input and we can stop.



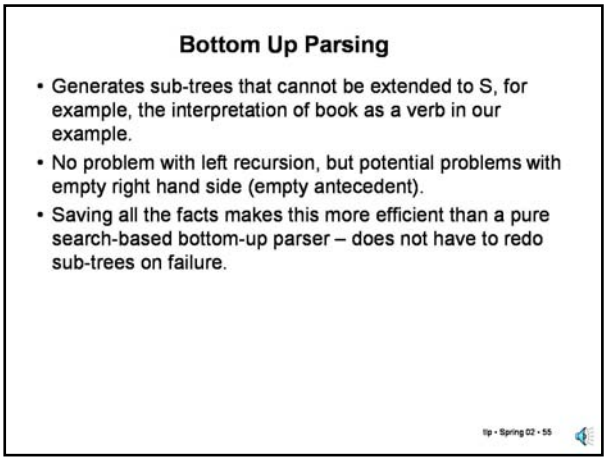


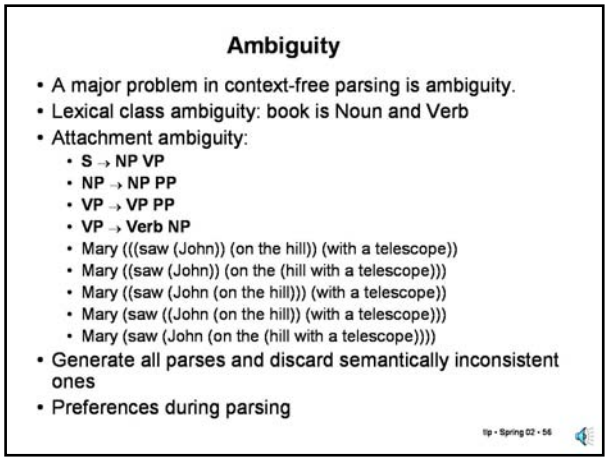
Slide 12.1.54

Note that (not surprisingly) we generated some facts that did not make it into our final structure.

Slide 12.1.55

Bottom-up parsing, like top-down parsing, generates wasted work in that it generates structures that cannot be extended to the final sentence structure. Note, however, that bottom-up parsing has no difficulty with left-branching rules, as top-down parsing did. Of course, rules with an empty right hand side can always be used, but this is not a fundamental problem if we require that triggering requires that a rule adds a new fact. In fact, by adding all the intermediate facts to the data base, we avoid some of the potential wasted work of a pure search-based bottom-up parser.





Slide 12.1.56

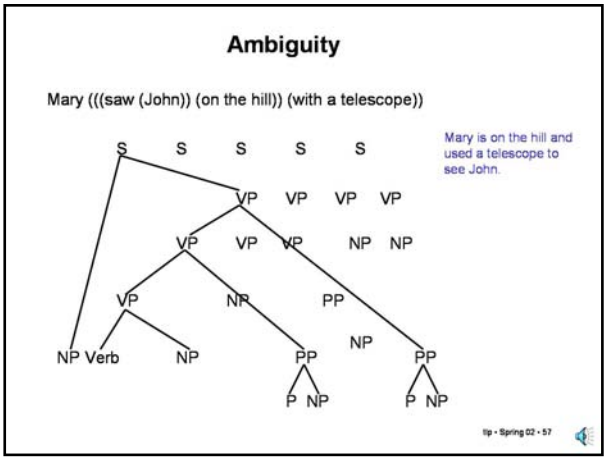
One of the key facts of natural language grammars is the presence of ambiguity of many types. We have already seen one simple example of **lexical ambiguity**, the fact that the word book is both a noun and a verb. There are many classic examples of this phenomenon, such as "Time flies like an arrow", where all of "time", "flies" and "like" are ambiguous lexical items. If you can't see the ambiguity, think about "time flies" as analogous to "fruit flies".

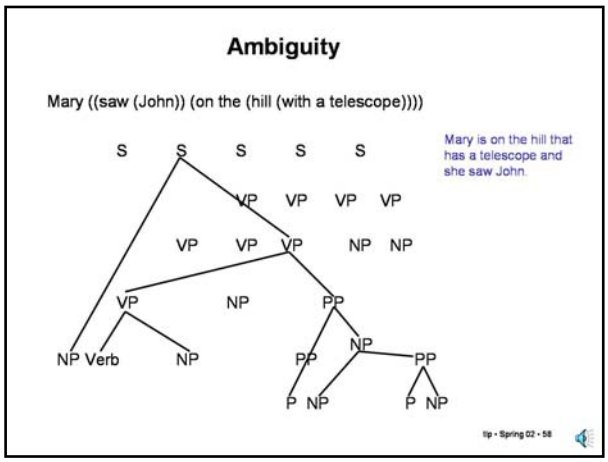
Perhaps a more troublesome form of ambiguity is known as **attachment ambiguity**. Consider the simple grammar shown here that allows prepositional phrases to attach both to VPs and NPs. So, the sentence "Mary saw John on the hill with a telescope" has five different structurally different parses, each with a somewhat different meaning (we'll look at them more carefully in a minute).

Basically we have two choices. One is to generate all the legal parses and let subsequent phases of the analysis sort them out or somehow to select one - possibly based on learned preferences based on examples. We will assume that we simply generate all legal parses.

Slide 12.1.57

Here are the various interpretations of our ambiguous sentence. In this one, both prepositional phrases are modifying the verb phrase. Thus, Mary is on the hill she used a telescope to see John.



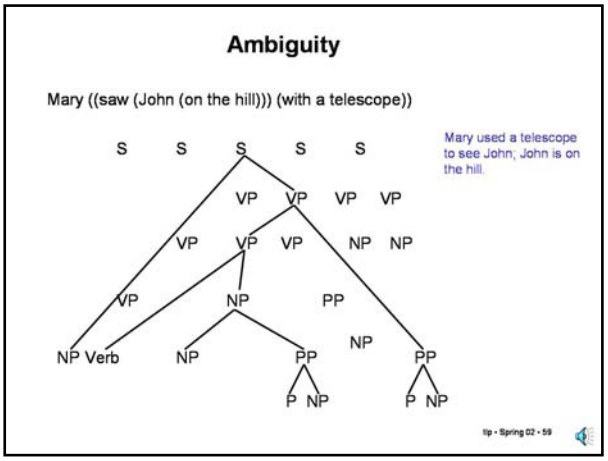


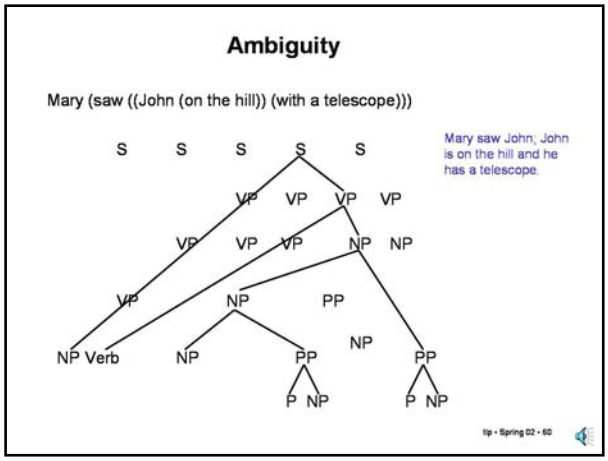
Slide 12.1.58

In this one, the telescope phrase has attached to the hill NP and so we are talking about a hill with a telescope. This whole phrase is modifying the verb phrase. Thus Mary is on the hill that has a telescope when she saw John.

Slide 12.1.59

In this one, the hill phrase is attached to John; this is clearer if you replace John with "the fool", so now Mary saw "the fool on the hill". She used a telescope for this, since that phrase is attached to the VP.





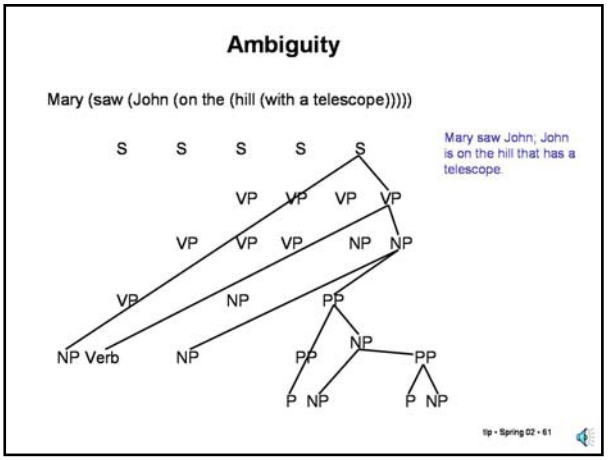
Slide 12.1.60

In this one, its the fool who is on the hill and who has the telescope that Mary saw.

Slide 12.1.61

Now its the fool who is on that hill with the telescope on it that Mary saw.

Note that the number of parses grows exponentially with the number of ambiguous prepositional phrases. This is a difficulty that only detailed knowledge of meaning and common usage can resolve.

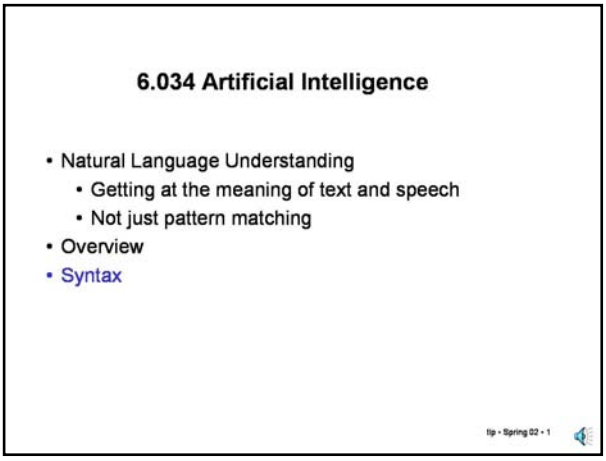


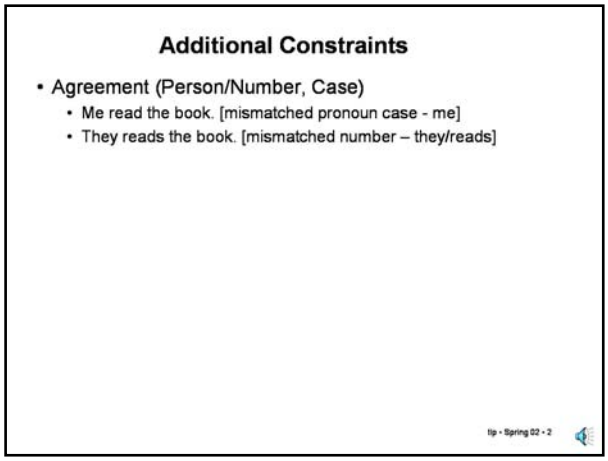
6.034 Notes: Section 12.2

Slide 12.2.1

In this section we continue looking at handling the syntax of natural languages.

Thus far we have been looking at very simple grammars that do not capture nearly any of the complexity of natural language. In this section we take a quick look at some more complex issues and introduce an extension to our simple grammar rules, which will prove exceedingly useful both for syntactic and semantic analysis.





Slide 12.2.2

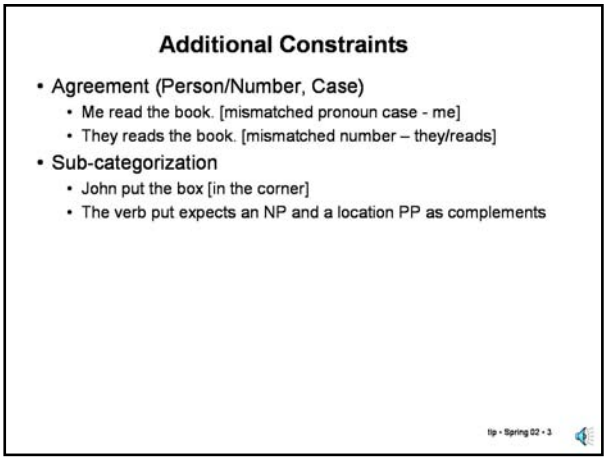
One important class of phenomena in natural language are agreement phenomena. For example, pronouns in a subject NP must be in the subjective case, such as, I, he, they, while pronouns in a direct object NP must be in the objective case, such as, me, him, them.

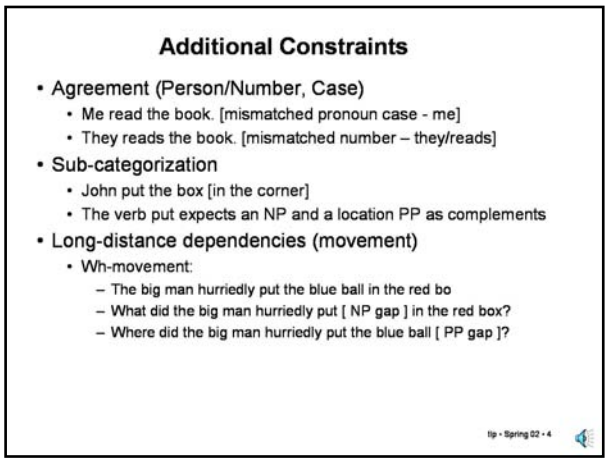
More interestingly, the person and number of the subject NP must match those of the verb. Some other languages, such as Spanish, require gender agreement as well. We will need some mechanism of capturing this type of agreement in our grammars.

Slide 12.2.3

Here is another form of agreement phenomena. Particular verbs requires a particular combination of phrases as complements. For example, the verb put expects an NP, indicating the object being put, and it expects a prepositional phrase indicating the location.

In general verbs can be sub-categorized by their expected complements, called their sub-categorization frame. We need to find some way of capturing these constraints.



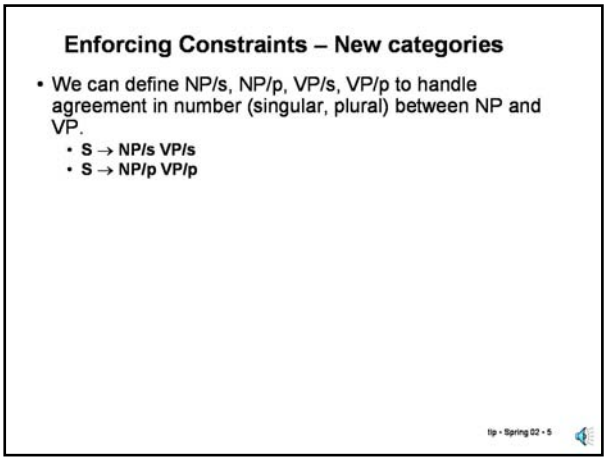


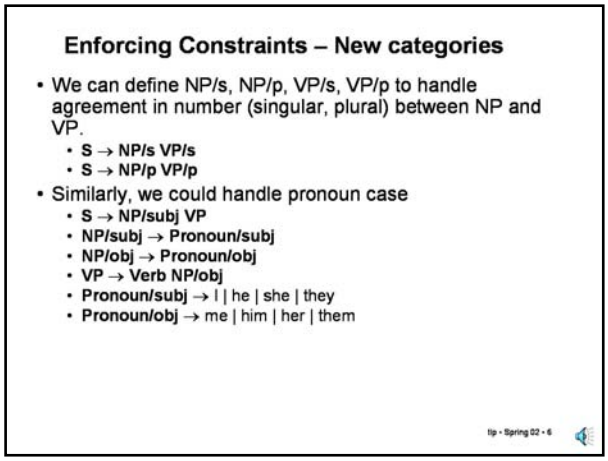
Slide 12.2.4

Another important class of phenomena can be understood in terms of the movement of phrases in the sentence. For example, we can think of a question as moving a phrase in the corresponding declarative sentence to the front of the sentence in the form of a wh-word, leaving a sort of "hole" or "gap" in the sentence where a noun phrase or prepositional phrase would have normally appeared. We will look at this type of sentence in more detail later.

Slide 12.2.5

There is one natural mechanism for enforcing agreement in context-free grammars, namely, to introduce new non-terminals - such a singular and plural noun phrases and verb phrases and then introduce rules that are specific to these "derived" classes.



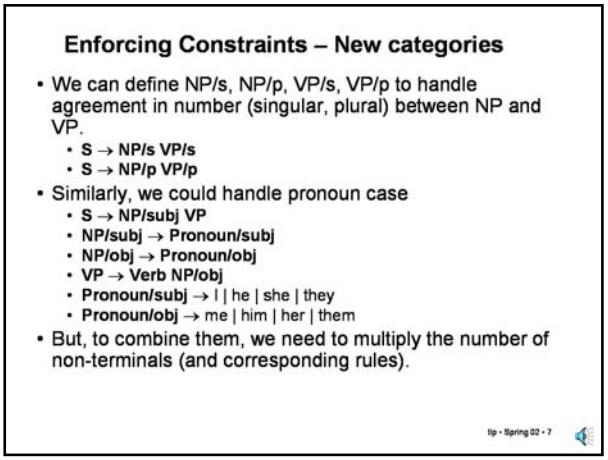


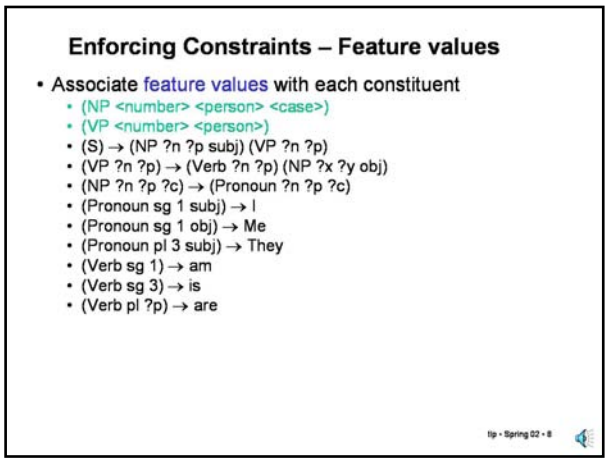
Slide 12.2.6

Note that we could extend this approach to handle the pronoun case agreement example we introduced earlier.

Slide 12.2.7

However, there is a substantial problem with this approach, namely the proliferation of non-terminals and the resulting proliferation of rules. Where we had a rule involving an NP before, we now need to have as many rules as there are variants of NP. Furthermore, the distinctions multiply. That is, if we want to tag each NP with two case values and two number values and 3 person values, we need 12 NP subclasses. This is not good...





Slide 12.2.8

An alternative approach is based on exploiting the unification mechanism that we have used in our theorem provers and rule-chaining systems. We can introduce variables to each of the non-terminals which will encode the values of a set of **features** of the constituent. These features, for example, can be number, person, and case (or anything else).

Now, we can enforce agreement of values within a rule by using the same variable name for these features - meaning that they have to match the same value. So, for example, the S rule here says that the number and person features of the NP have to match those of the VP. We also constrain the value of the case feature in the subject NP to be "subj".

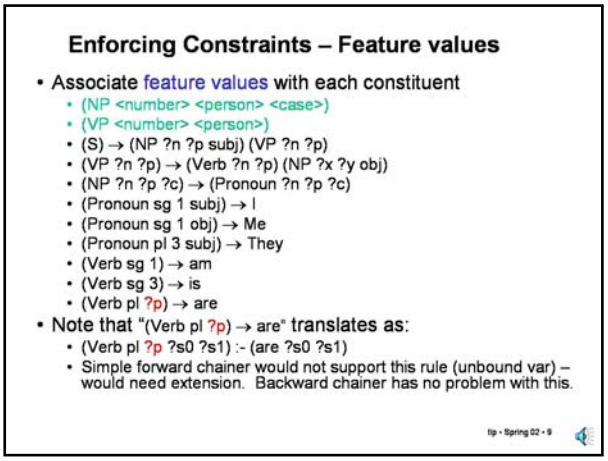
In the VP rule, note that the number and person of the verb does not need to agree with that of the direct object NP, whose case is restricted to be "obj".

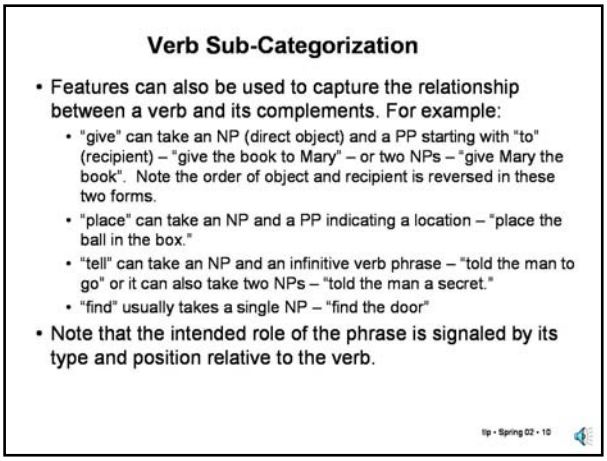
Most of the remaining rules encode the values for these features for the individual words.

Slide 12.2.9

The last rule indicates that the verb "are" is plural and agrees with any person subject. We do this by introducing a variable instead of a constant value. This is straightforward except that in the past we have restricted our forward chainer to dealing with assertions that are "ground", that is, that involve no variables. To use this rule in a forward-chaining style, we would have to relax that condition and operate more like the resolution theorem prover, in which the database contains assertions which may contain variables.

In fact, we could just use the resolution theorem prover with a particular set of preferences for the order of doing resolutions which would emulate the performance of the forward chainer.





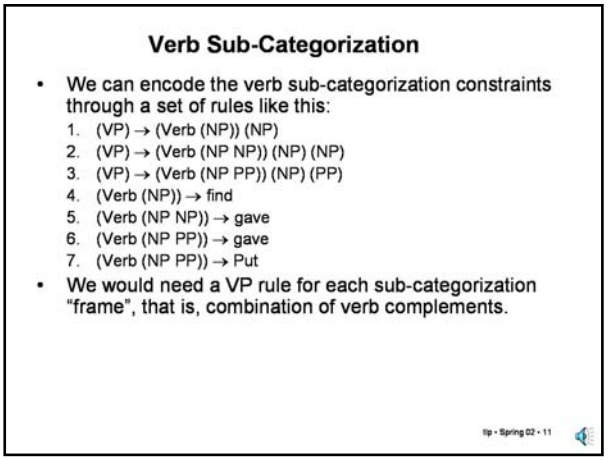
Slide 12.2.10

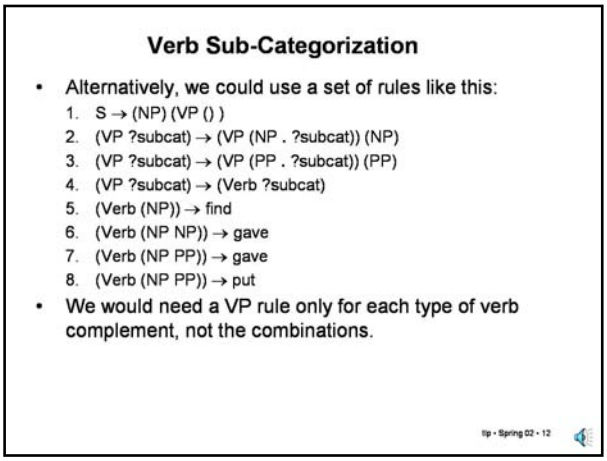
Let's look now at verb sub-categorization. We have seen that verbs have one or more particular combinations of complement phrases that are required to be present in a legal sentence. Furthermore, the role that the phrase plays in the meaning of the sentence is determined by its type and position relative to the verb.

We will see that we can use feature variables to capture this.

Slide 12.2.11

One simple approach we can use is simply to introduce a rule for each combination of complement phrases. The Verb would indicate this complex feature and only the rule matching the appropriate feature value would be triggered. However, we would need a large number of rules, since there are many different such combinations of phrases possible. So, a more economical approach would be desirable.





Slide 12.2.12

Here we see an alternative implementation that only requires one rule per complement phrase type (as opposed to combinations of such types). The basic idea is to use the rules to implement a recursive process for scanning down the list of expected phrases.

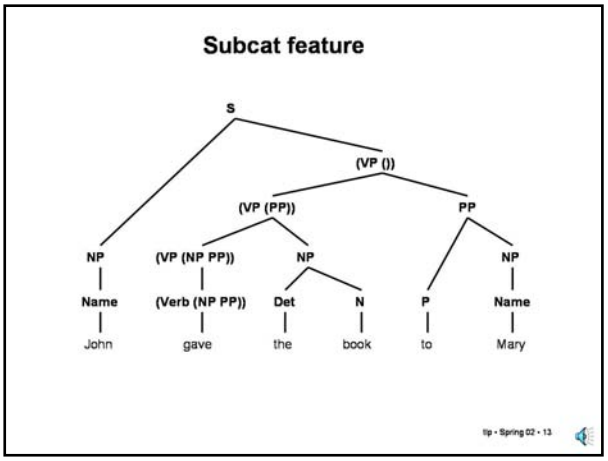
In fact, this set of rules can be read like a Scheme program. Rule 1 says that if the subcat list is empty then we do not expect to see any phrases following the VP, just the end of the sentence. So, this rule will generate the top-level structure of the sentence.

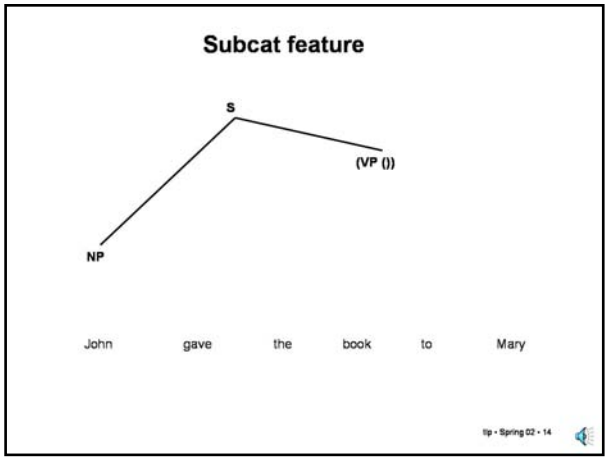
Rules 2 and 3 handle the cases of a noun phrase or propositional phrase expected after the verb phrase. If you look closely, these rules are a bit strange because they are rewriting a simpler problem, a verb phrase with a subcat list, into what appears to be a more complex phrase, namely another verb phrase with a longer subcat list which is followed by a phrase of the appropriate type. Imagine that the subcat list were null, then rule 2 expands such a VP into anther VP where the subcat list contains an NP and this VP is followed by an actual NP phrase. Rule 3 is similar but for prepositional phrases.

The idea of these rules is that they will expand the null subcat list into a longer list, at each point requiring that we find the corresponding type of phrase in the input. The base case that terminates the recursion is rule 5, which requires finding a verb in the input with the matching subcat list. An example should make this a bit clearer.

Slide 12.2.13

Here's an example of using this grammar fragment. You can see the recursion on the VP argument (the subcat feature) in the three nested VPs, ultimately matching a Verb with the right sub-categorization frame. Let's look in detail at how this happens.



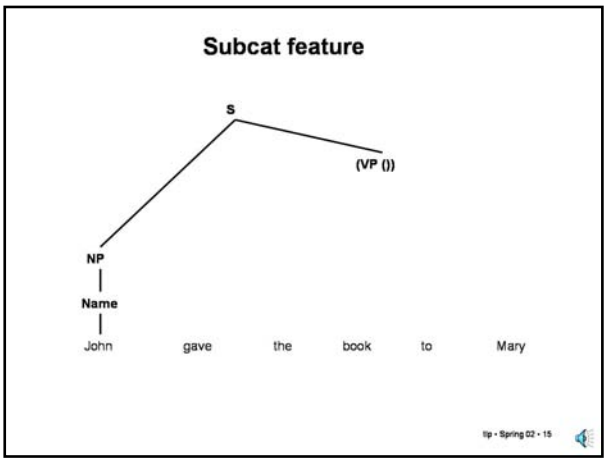


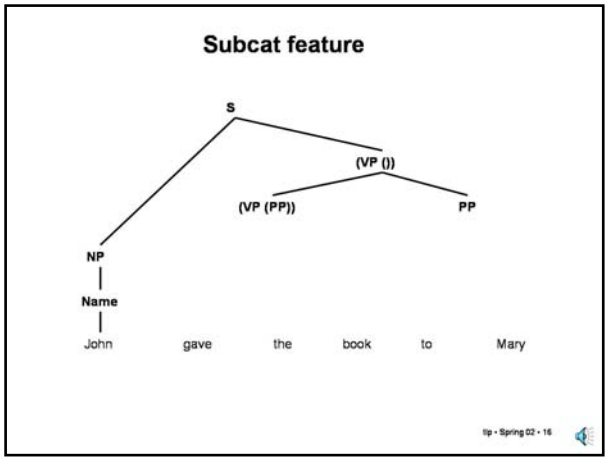
Slide 12.2.14

We start with the top-level S rule, which creates an NP subgoal and a VP subgoal with the ?subcat feature bound to the empty list.

Slide 12.2.15

The NP rule involving a name succeeds with the first input word: John.



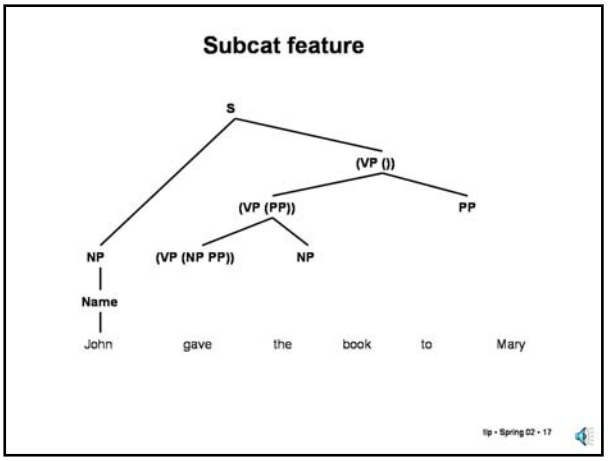


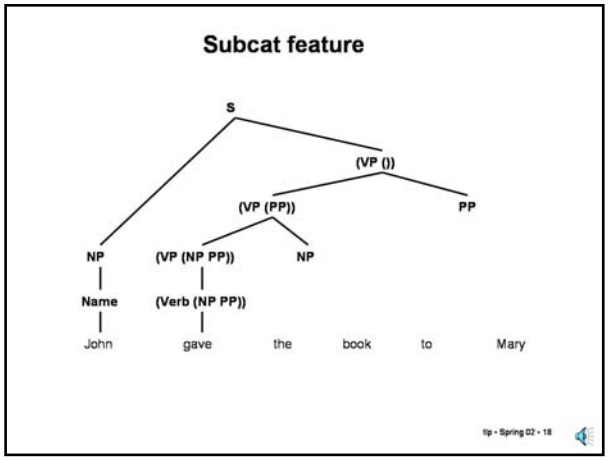
Slide 12.2.16

In practice, we would have to try each of the VP rules in order until we found the one that worked to parse the sentence. Here, we have just picked the correct rule, which says that the VP will end with a prepositional phrase PP. This is rule 3 in the grammar. Note that this involves binding the ?subcat variable to (). Note that this creates a new VP subgoal with ?subcat bound to (PP).

Slide 12.2.17

We now pick rule 2 with ?subcat bound to (PP). This rule will look for an NP in front of the PP and create a new VP subgoal with ?subcat bound to (NP PP).



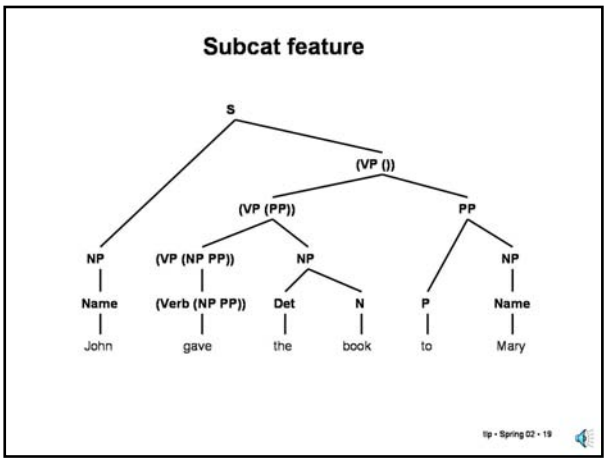


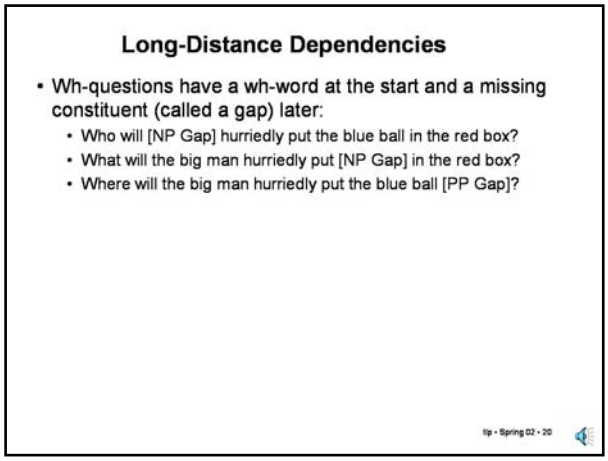
Slide 12.2.18

We now need to use rule 4, which proceeds to find a Verb with ?subcat bound to (NP PP), that is, a verb that accepts a direct object and a prepositional phrase as complements, for example, the verb "gave".

Slide 12.2.19

The rest of the parse of the sentence can proceed as normal.





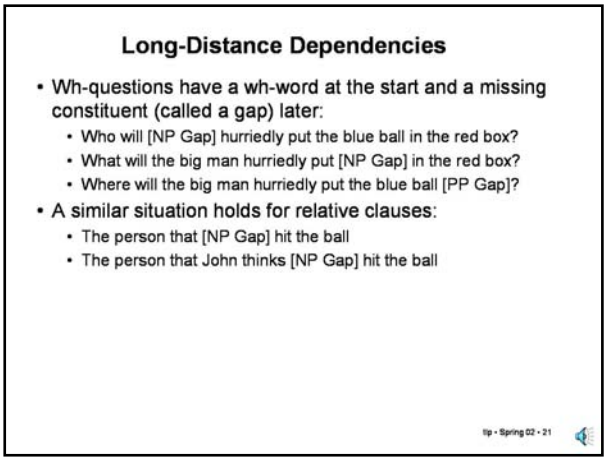
Slide 12.2.20

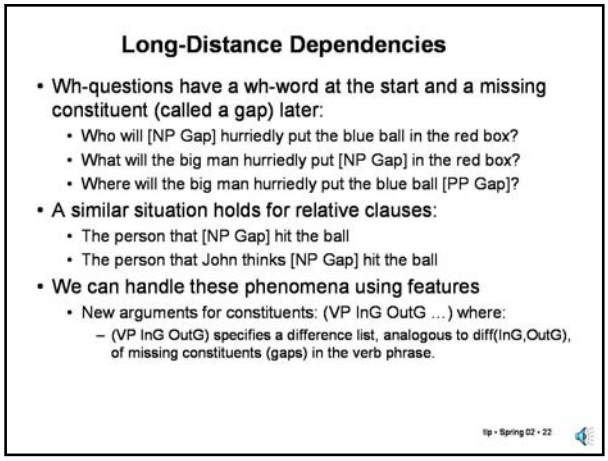
Let's consider how to parse wh-questions, which have a wh-word (what, where, who, when) at the start and a missing constituent phrase, an NP or a PP in the rest of the sentence. The missing phrase is called a **gap**. In these questions, the "will" is followed by a sentence that follows the usual rules for sentences except that in each case, the sentence is missing a phrase, indicated by the brackets.

We would like to parse these sentences without having to define a special grammar to handle missing constituents. We don't have to define a new sentence grammar that allows dropping the subject NP and another one that allows dropping an object NP or an object PP. Instead, we would like to generalize our rules for declarative sentences to handle this situation.

Slide 12.2.21

The same can be said about a relative clause, which is basically a sentence with a missing NP (which refers to the head noun).





Slide 12.2.22

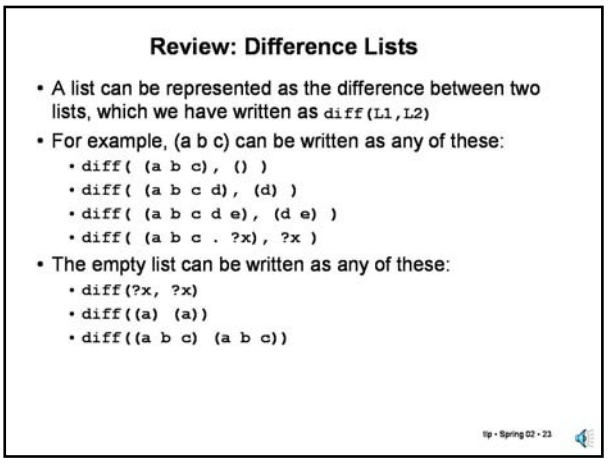
We can also handle these missing constituents by using feature variables. In particular, we will add two new arguments to those constituents that can have a missing phrase, that is, can accpet a gap. The first feature represents the beginning of a list of gaps and the second represents the end of the list. It is the difference between these two lists that encodes whether as gap has been used. So, if the first value is equal to the second, then no gap has been used. We will see how this work in more detail in a minute, but first let's review difference lists.

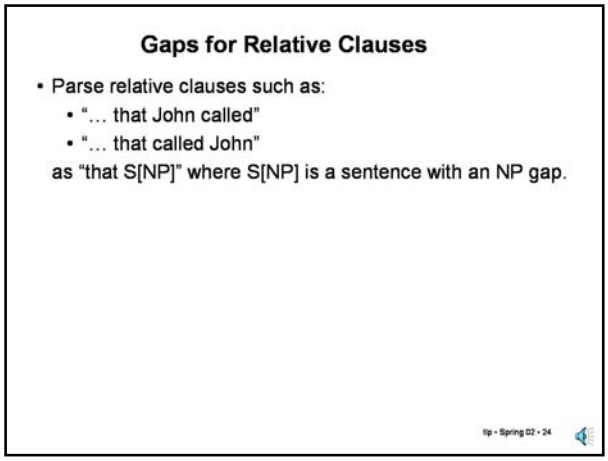
Slide 12.2.23

We saw when we studied logic programs that we could manipulate lists using a representation called **difference lists**. You can see some examples of this representation of a simple list with three elements here. The basic idea is that we can represent a list by two variable, one bound to the beginning of the list and the other to the end of the list. Note that if the first and second variables are bound to the same value, then this represents an empty list.

In the grammars we will be dealing with in this chapter, we will only need to represent lists of at most length one.

Also, note, that the crucial thing is having two variable values, the symbol diff doesn't actually do anything. In particular, we are **not** "calling" a function called diff. It's just a marker used during unification to indicate the type of the variables.





Slide 12.2.24

Let's look at a piece of grammar in detail so that we can understand how gaps are treated. We will look at a piece of the grammar for relative clauses. Later, we look at a bigger piece of this grammar.

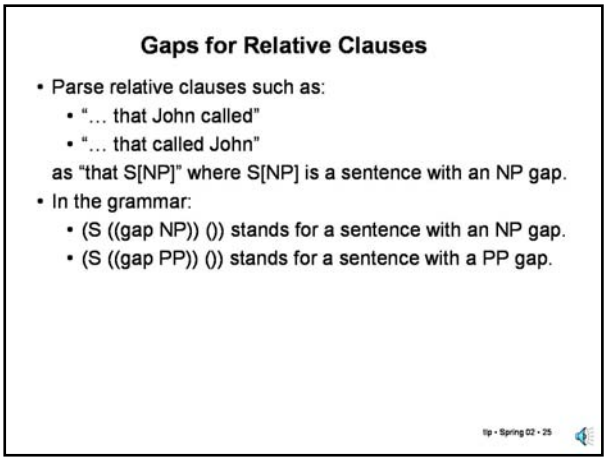
The key idea, as we've seen before, is that a relative clause is a sentence that is missing a noun phrase, maybe the subject noun phrase or an object noun phrase. The examples shown here illustrate a missing object NP, as in, "John called the man" becoming "that John called". Or, a missing subject NP, as in, "The man called John" becoming "that called John".

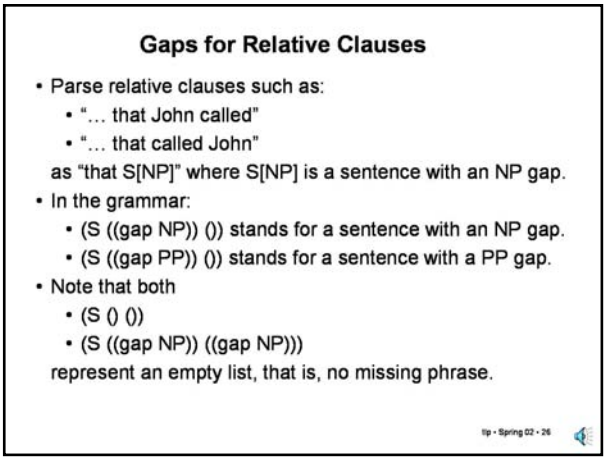
Slide 12.2.25

In our grammar, we are going to add two variables to the sentence literal, which will encode a difference list of gaps. This literal behaves exactly as a difference list, even though we don't show the "diff" symbol. The examples we show here are representing a list of one element. The one element is a list (gap NP) or (gap PP). This convention is arbitrary, we could have made the elements of this list be "foo" or "bar" as long as we used them consistently in all the rules. We have chosen to use a mnemonic element to indicate that it is a gap and the type of the missing component.

Now, if we want to have a rule that says that a relative clause is the word "that" followed by a sentence with a missing NP, we can do that by using this

```
(RelClause) :- "that" (S ((gap NP)) () )
```





Slide 12.2.26

And, just as we saw with other difference lists, if both gap variables are equal then this represents an empty list, meaning that there is no missing component. That is, the sentence needs to be complete to be parsed. In this way, we can get the behavior we had before we introduced any gap variables.

Slide 12.2.27

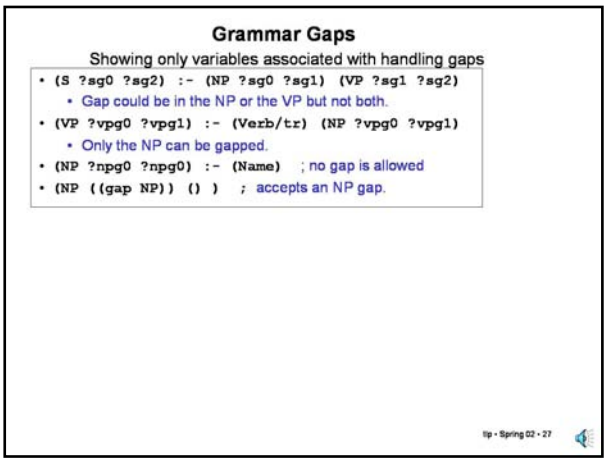
Here is a small, very simplified, grammar fragment for a sentence that allows a single NP gap. We have added two gap variables to each sentence (S), noun phrase (NP) and verb phrase (VP) literal.

The first rule has the same structure of the append logic program that we saw in the last chapter. It says that the sentence gap list is the append of the NP's gap list and the VP's gap list. This basically enforces conservation of gaps. So, if we want a sentence with one gap, we can't have a gap both in the NP and the VP. We'll see this work in an example.

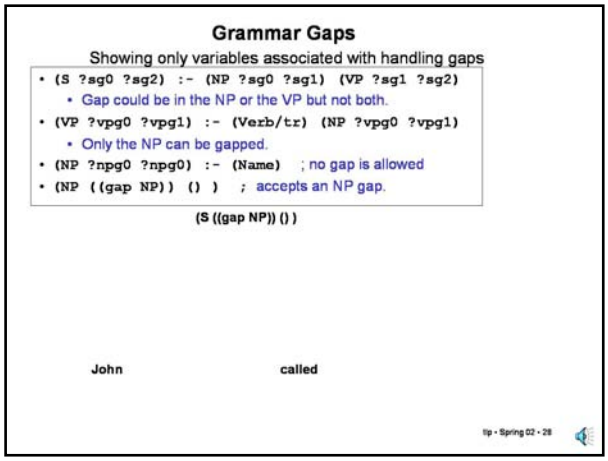
The second rule shows that if there is a gap in the VP, it must be the object NP. The third rule is for a non-gapped NP, which in this very simple grammar can only be a name.

The last rule is the one that is actually used to "recognize" a missing NP. Note that this rule has no antecedent and so can be used without using any input from the sentence. However, it will only be used where a gap is allowed since the gap variables in the rule need to match those in the goal.

We have left out the rules for specific words, for example, that John is a name or that called is a transitive verb.



Note that, in general, there will be other variables associated with the grammar literals that we show here, for example, to enforce number agreement. We are not showing all the variables, only the gap variables, so as to keep the slides simpler.



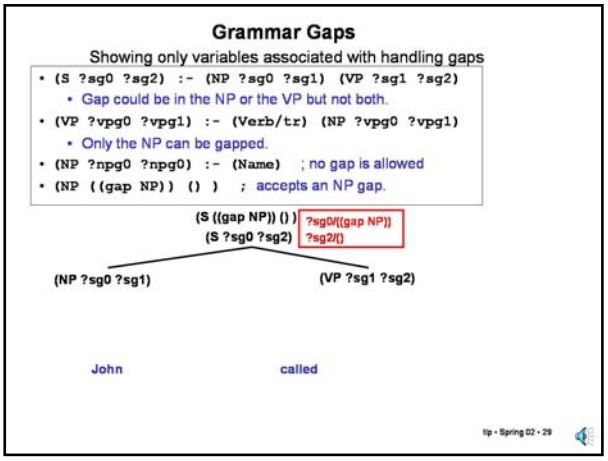
Slide 12.2.28

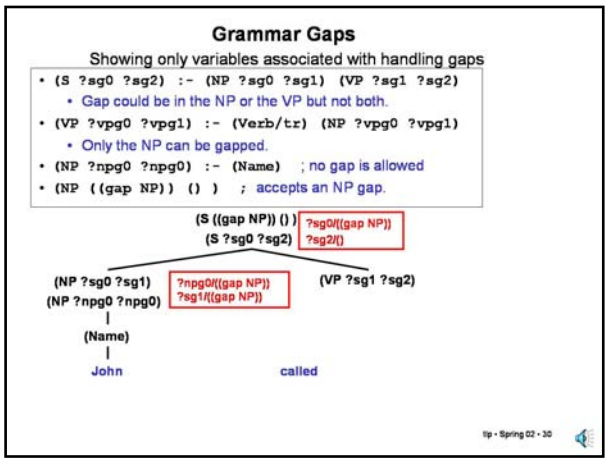
Let's see how this grammar could be used to parse a sentence with a missing object NP, such as "John called" that would come up in the relative clause "that John called".

The goal would be to find a sentence with an NP gap and so would be the S literal shown here.

Slide 12.2.29

This goal would match the consequent of the first rule and would match the gap variables of the S literal in the process. Note that the ?sg1 variable is unbound. It is binding this variable that will determine whether the gap is in the NP or in the VP. Binding ?sg1 to ((gap NP)) would mean that the gap is in the VP, since then the gap variables in the subject NP would be equal, meaning no gap there. If ?sg1 is bound to () then the gap would be in the subject NP.



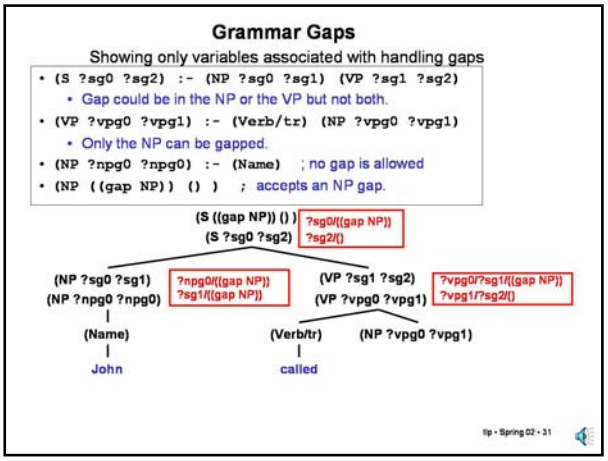


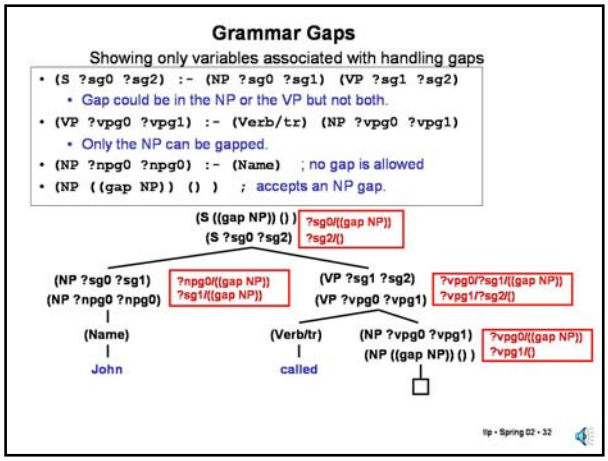
Slide 12.2.30

Using the first NP rule, we can succesfully parse "John". In the process, ?sg1 would have to be bound to the same value as ?sg0 for them both to unify with ?npg0. At this point, we've committed for the gap to be in the verb phrase in order for the whole parse to be succesful.

Slide 12.2.31

Now we use the VP rule. Note that at this point, the gap variables are already bound from our previous unifications.





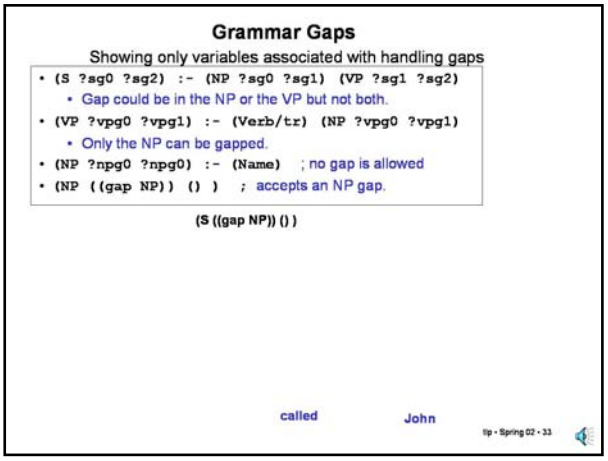
Slide 12.2.32

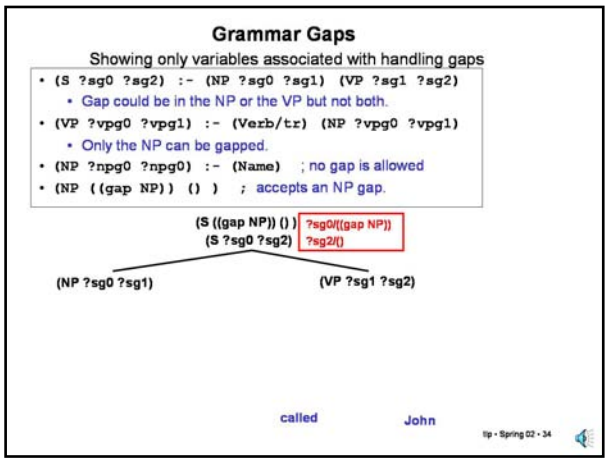
Now we use the NP rule that accepts the gapped (that is, missing) NP. This rule is acceptable since the bindings of the gap variables are consistent with the rule.

So, we have succesfully parsed a sentence that has the object NP missing, as required.

Slide 12.2.33

Now, let's look at what happens if it is the subject NP that is missing from the sentence, such as would happen in the relative clause "that called John". We start with the same goal, to parse a sentence with an NP gap.



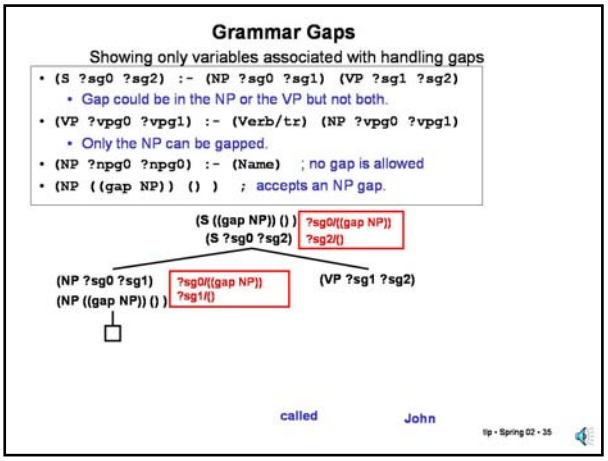


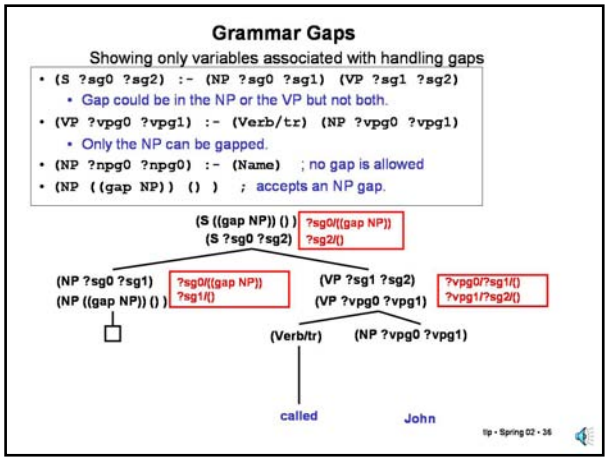
Slide 12.2.34

As before, we use the top-level sentence rule, which binds the gap variables, ?sg0 and ?sg2, leaving ?sg1 unbound.

Slide 12.2.35

Now, we need to parse the subject NP. The parse would try the first NP rule, that would require finding a name in the sentence, but that would fail. Then, we would try the gapped NP rule, which succeeds and binds ?sg1 to (). At this point, we've committed to the gap being in the subject NP and not the VP.





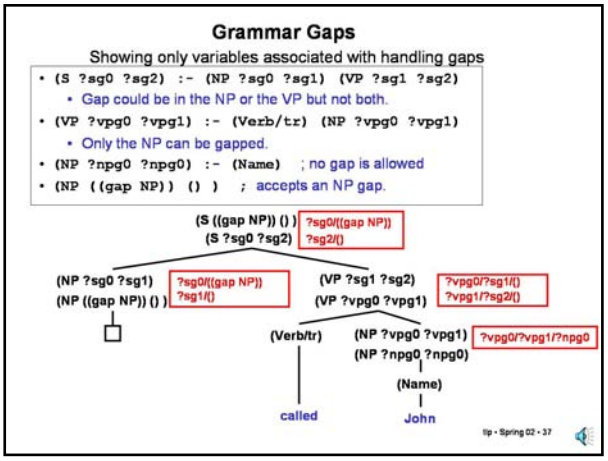
Slide 12.2.36

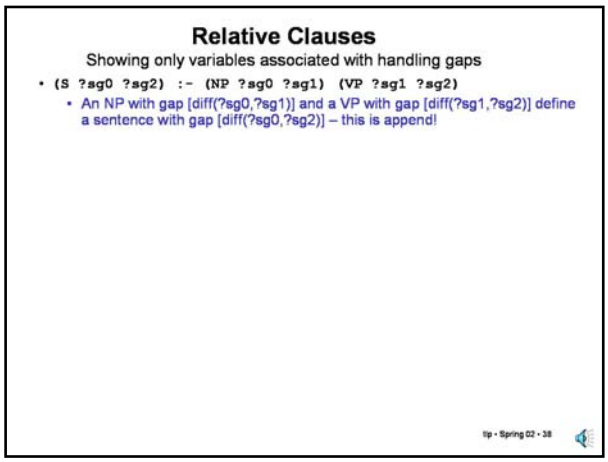
The VP rule would now be used and then the Verb rule would accept "called".

Slide 12.2.37

The NP rule would then be used. Note that the gap variables are already both bound to (), so there is no problem there. The Name rule would then recognize John.

So, we see that using the same set of rules that we would use to parse a normal sentence, we can parse sentences missing an NP, either in the subject or object position. In general, we can arrange to handle gaps for any type of phrase.



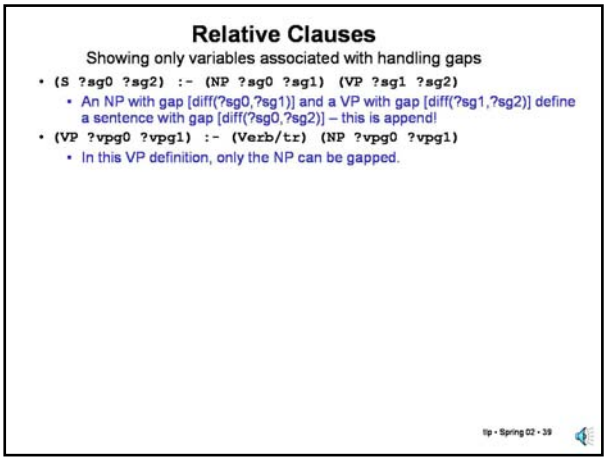


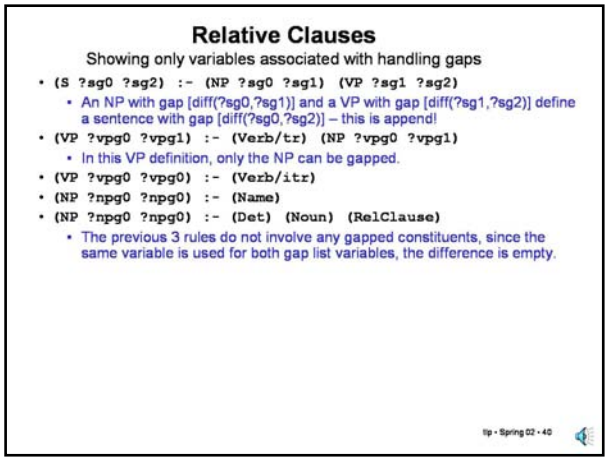
Slide 12.2.38

So, let's look at a more complete grammar for relative clauses using gaps. The Sentence rule simply indicates that the gap list is distributed in some way among the NP and VP constituents.

Slide 12.2.39

The transitive VP rule indicates that only the NP can be gapped, we can't drop the Verb.



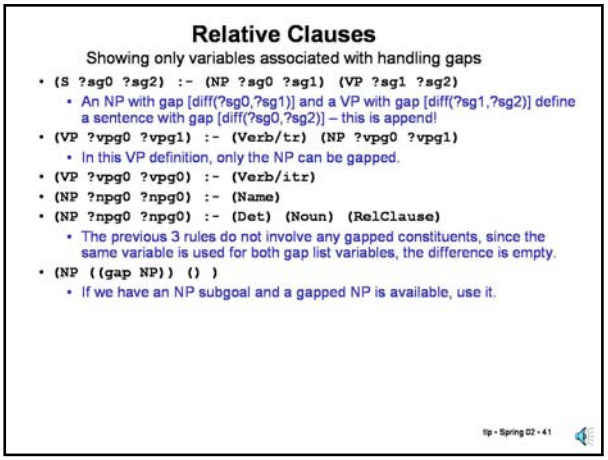


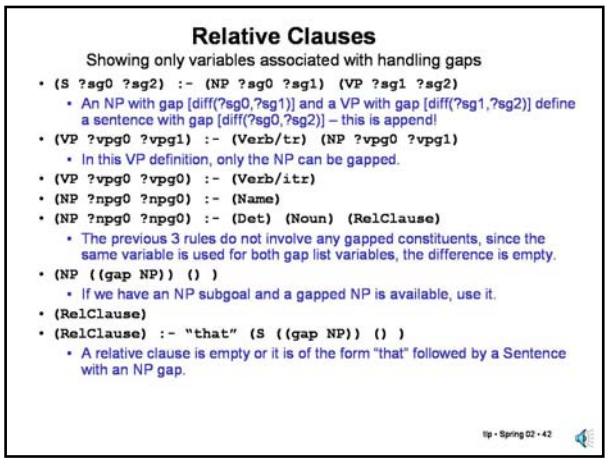
Slide 12.2.40

The next three rules say that there can be no gapped constituents since the first and second gap features are constrained to be the same, since the same variable name is used for both.

Slide 12.2.41

Now, this is an important rule. This says that if we use a gapped NP in this constituent we don't need any additional input in order to succeed in parsing an NP. This rule "fills the gap".



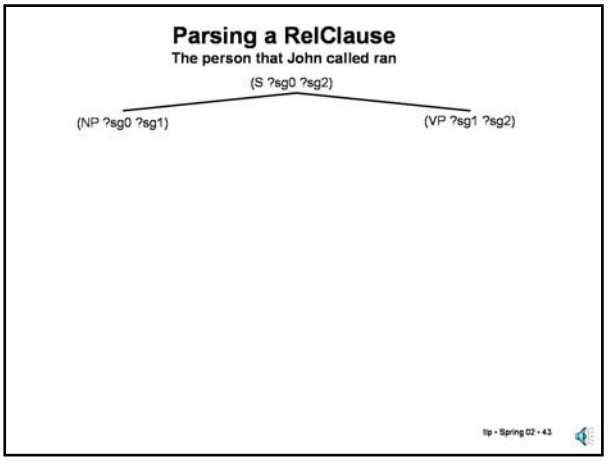


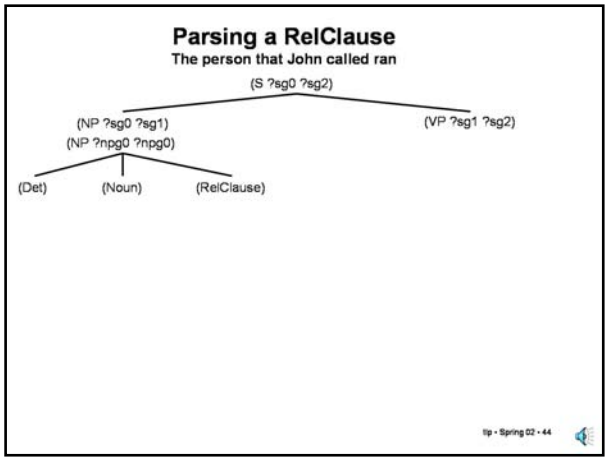
Slide 12.2.42

Finally, we get the the definition of the relative clause. The first rule just says that the RelClause is optional. The second rule is the key one, it says that a RelClause is composed of the word "that" followed by a sentence with a missing NP and it provides the NP to be used to fill the gap as necessary while parsing S.

Slide 12.2.43

Let's see how we can parse the sentence "The person John called ran". We start with the S rule.



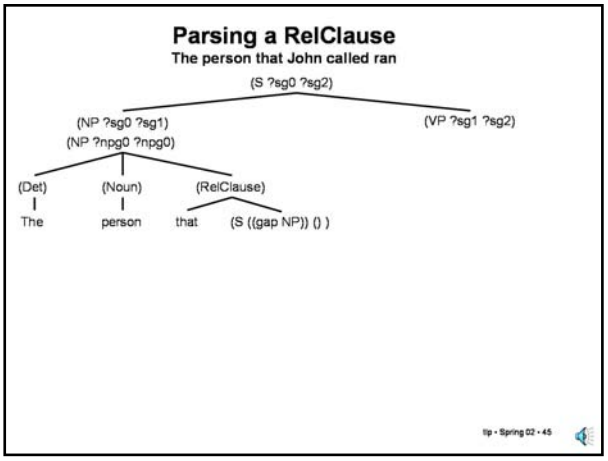


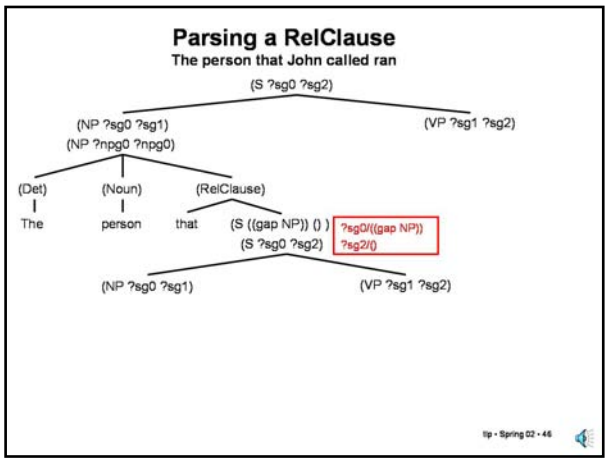
Slide 12.2.44

We then use the NP rule which generates three subgoals.

Slide 12.2.45

We proceed to parse the Determiner and the noun, the RelClause then sets the subgoal of parsing a sentence with a missing NP.



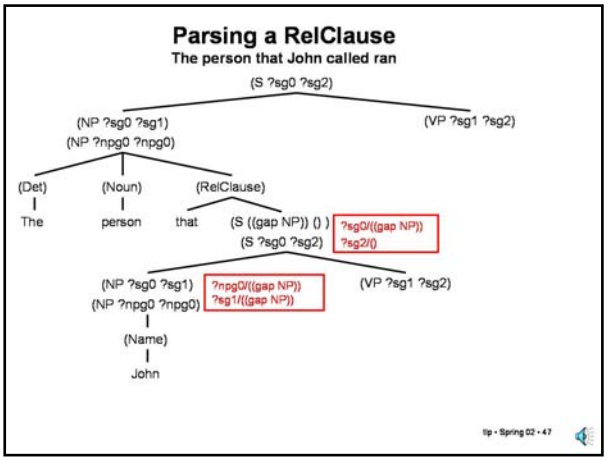


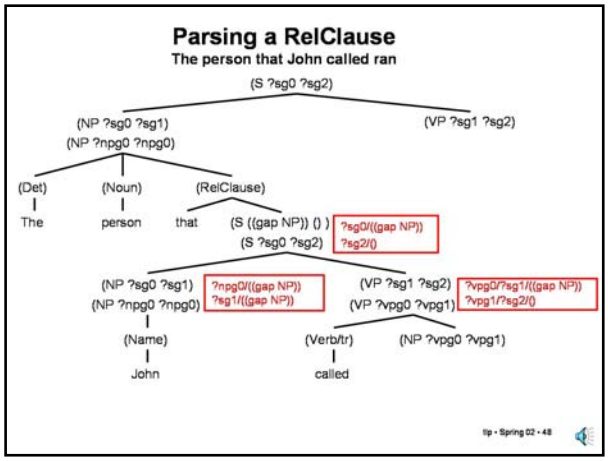
Slide 12.2.46

We use our S rule again, but now we have ?sg0 bound to ((gap NP)) and ?sg2 is bound to the empty list. We now proceed with the subject NP for this embedded sentence.

Slide 12.2.47

Note that we end up using the Name rule in which the gap features are constrained to be equal. So that means that the gap NP is not used here. As a result of this match, we have that ?sg1 is now equal to ((gap NP)).





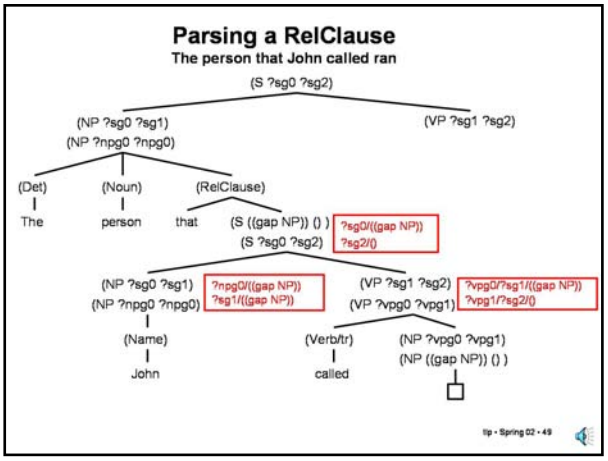
Slide 12.2.48

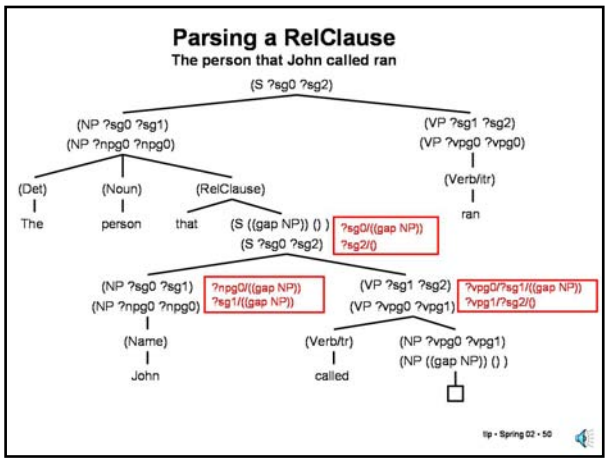
Now, we proceed to parse the VP of the embedded sentence, noting that ?vpg0 is ((gap NP)) and the ?vpg1 is the empty list. This means that we expect to use the gap in parsing the VP.

We proceed to parse the Verb - "called".

Slide 12.2.49

Now, we need an NP but we want to use a gap NP, so we succeed with no input.





Slide 12.2.50

We finish up by parsing the VP of the top-level sentence using the remaining word, the verb "ran".

Which is kind of cool...