(1983/2017). In A. Bunger (Ed.)., *Special volume: Reissue of Innovations in Linguistic Education Volume 3. IULC Working Papers in Linguistics*, Vol. 17, No. 4.

NEW TOOLS FOR OLD TASKS: AN INTRODUCTORY COURSE IN COMPUTATIONAL LINGUISTICS

Karen Jensen

Computer Sciences Department
IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

1. Introduction

The course on which this paper is based was developed and taught at Hofstra University, Hempstead, Long Island. Its primary innovation is the introduction of NLP (Natural Language Processor; Heidorn 1972, 1975), a computer programming language that is especially designed for handling natural (human) languages. Because the statements of NLP are phrase structure rules, the language is particularly congenial to linguists and linguistics students. It allows for explicit and direct reflection of linguistic intuitions, and it implements them with the speed and processing power of the computer.

This introductory curriculum has twin goals:

(a) to further the study of linguistics and natural language, by giving its students a new tool which will demand precision and encourage insight;

(b) to further computational studies, by developing programs that can do interesting and productive things with natural language.

The course was centered around the study of syntax. However, NLP could equally well be used as a vehicle for a course in semantics, morphology, phonology, or any other rule-governed aspect of the human language system.

Key to the success of this course is the realization that programming is not alien to linguists, nor is the study of grammar alien to computer scientists--although neither group may be conscious of these facts. What a linguist does, when he writes or thinks about rules of grammar, is, essentially, to build a subroutine designed to handle a small part of the mental program that processes the language he is working with. Part of what a computer scientist does, when he learns a programming language, is to study grammar. It is easier for linguists

and computer scientists to recognize their kinship when they are provided with a derivational link, such as NLP.

This paper first gives a brief overview of NLP, and then a series of four graduated tasks designed to show how a generative (explicit) study of language can be encouraged by working out problems on a computer. Tasks such as these were used in the introductory course at Hofstra. Finally, two appendices present a small computational grammar and a brief outline for a 14-week semester course on English syntax and NLP. One small caveat should be inserted here: it is impossible, in this short paper, to do a thorough job of teaching NLP. Therefore some unclarities in the exposition are bound to exist. Anyone who wants further explanation is invited to write to the author.

2. Overview of NLP

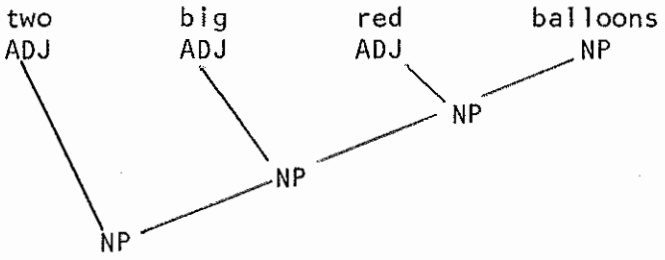
Communication is generally understood to be a process of sending and receiving information. Corresponding to this natural division of the communication process, NLP has two different kinds of rules: rules for decoding (receiving; inputting; parsing) and rules for encoding (sending; outputting; producing).

A simplified NLP DECODING rule looks like this:

(1) ADJ NP --> NP

This rule will take a string of adjectives preceding a noun phrase, and combine the adjectives one by one with the noun phrase to create a larger segment of type NP. Single nouns (like *balloons*) can also be called noun phrases, so the phrase *two big red balloons* will be decoded by successive application of rule (1) as follows:

(2)



Notice that successive operations of rule (1) lead to the building of a parse tree, which defines a syntactic structure for the phrase. The whole purpose of decoding rules is to do just that: to take an incoming string of lexical items and assign structure to them.

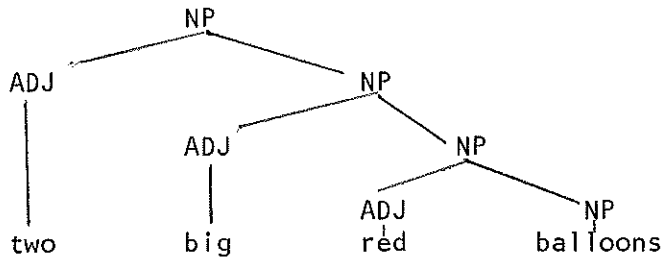
NLP decoding rules work from the bottom up and in a parallel process, which means that they look at the input string as a whole before constructing a final parse tree; that all rules which might conceivably apply to a string are applied; and that all possible structures, resulting from the application of any rule(s), are held in readiness until the final tree has been constructed.

A typical NLP ENCODING rule looks like this:

(3) NP --> ADJ NP

This rule will take a noun phrase segment type and produce from it a string of two elements, an adjective followed by a noun phrase. If necessary, the same rule can apply to a given segment type more than once:

(4)



As can be seen by comparing (2) and (4), encoding and decoding rules are able to build equivalent structures. But their purposes are very different. Decoding rules take strings of lexical items and turn them into segment types (phrases, clauses, etc); encoding rules take segment types and turn them into strings of lexical items. (The term SEGMENT TYPE is intended to be vague here. It can encompass syntactic, phonological, semantic, or any kind of information being worked with.)

Whereas decoding rules work bottom-up and in parallel, encoding rules work top down and in serial, which means that they start with a given segment type; apply to it the first listed rule that fits the segment description; ignore any other later rules that might have been applicable to that segment; and produce strings of lexical items from left to right.

In the programming tasks that follow, only ENCODING rules will be used. Encoding rules look more like the phrase-structure rules that linguists are familiar with; they allow experimentation with rule ordering; and they are easier for some students to comprehend since they produce recognizable strings of text, rather than abstract structures.

There is so far no NLP textbook. In choosing the text for this course, the teacher should use whatever introductory syntax book he is comfortable with. However, that book should lend itself to the computational approach, which is explicit and rigorous. In what follows, we will assume the use of one such, by now fairly standard, syntax text: Akmajian and Heny 1975.

3. The Random Generation of Noun Phrases Using Context-free Phrase Structure Rules

Akmajian and Heny first discuss noun phrases on page 26, presenting single groupings like *a baby* and *those three sheep*. As an early programming exercise, it is interesting to try to write the NLP rules that would produce such strings. One way to begin would be:

```
(5)  NP    -->  DET  ADJ  NOUN
     DET   -->  #  THOSE
     ADJ   -->  #  THREE
     NOUN  -->  #  SHEEP
```

(The '#' mark forces a space in the output, and can be used to provide for separation between words.)

But this set of rules will only produce the single string, *those three sheep*. Even if we add many more rules for DET, ADJ and NOUN, introducing more terminal symbols, our output will be limited because NLP encoding rules choose only the FIRST applicable rule of any segment type.

To avoid this problem, NLP provides for random selection among those rules that have identical left-hand sides. If we add more rules for determiners, adjectives, and nouns, we can get many noun phrases consisting of three words: *these good boys, the tall girl, a green idea,* and so on.

But how about noun phrases like *a baby*? In such strings there is no adjective intervening between determiner and noun. Linguists allow for optional elements by using parentheses or brackets, e.g.: NP --> Det (Adj) Noun. But NLP does not use these abbreviatory conventions. There are two ways to provide for optionality in NLP rules. One is to write a second rule, omitting the optional segment type:

```
(6)  a)   NP  -->  DET  ADJ  NOUN
     b)   NP  -->  DET  NOUN
```

The other is to use the segment type 'NULL'. The NULL segment is interpreted by NLP to mean nothing--not even a space:

(7) a) NP --> DET ADJ NOUN
b) ADJ --> # THREE
c) ADJ --> NULL

Now we have rules to generate NPs like *those three sheep* and *a baby*. Fine, but hardly a sufficient sampling of possible NPs in English. What rules are missing? One fairly obvious weakness of the rules in (5-7) is that none of them can handle theoretically infinite strings of words. We know, for instance, that there is no definable bound on the number of adjectives that can premodify a noun in English. Yet 'DET ADJ NOUN' limits us to one determiner and one adjective.

The way around this problem, of course, is to make use of recursion. Consider these rules for adjective phrases (AP):

(8) a) AP --> ADJ AP
b) AP --> ADJ

The recursive rule (8a) will allow for an infinitely long string of adjectives. In fact, (8a) is unstoppable, and will loop forever, unless a rule like (8b) is present as a possible choice for rewriting AP. Putting a computer into an infinite loop is not recommended as a homework exercise; but it certainly is a graphic demonstration of the power of recursion.

Through careful description and constant questioning, students can be brought to understand both how complex language is and how it can be effectively modeled. Noun phrases are fertile ground for a beginning task of this sort. Why is it, for example, that *the two big red balloons* is far better than *the red big two balloons*? The commonly accepted phrase structure rule, using the asterisk to denote an indefinite number of adjectives:

(9) NP --> Det (Adj)* Noun

does not address this question.

Consider the following set of NLP encoding rules (and compare them with Chomsky 1957:26):

(10) Encoding Rules

```
a)   NP    -->  DET  NP1
b)   NP    -->  NP1
c)   NP1   -->  NUM  AP  COL  NOUN
d)   AP    -->  ADJ  AP
e)   AP    -->  ADJ

f)   DET   -->  #  A
g)   DET   -->  #  THE
h)   DET   -->  #  THOSE
i)   NUM   -->  NULL
j)   NUM   -->  #  ONE
k)   NUM   -->  #  TWO
l)   NUM   -->  #  THREE
m)   ADJ   -->  #  BIG
n)   ADJ   -->  #  SMALL
o)   ADJ   -->  #  COLORLESS
p)   COL   -->  #  RED
q)   COL   -->  #  GREEN
r)   COL   -->  #  HELIOTROPE
s)   NOUN  -->  #  BALLOON
t)   NOUN  -->  #  IDEAS
u)   NOUN  -->  #  SHEEP
```

The writing of this encoding rule set, or one very like it, can be assigned to beginning students as an early step in elaborating the interior structure of the simple English noun phrase. When run on the computer, these rules will randomly produce such NPs as:

one small balloon
a one big small heliotrope sheep
**the three big colorless small big small small small colorless small big small heliotrope balloon*
colorless green ideas

and can serve as fuel for discussion of such questions as:

'What is linguistic generality?' ('How general is this rule set?')
'How should lexical items be introduced?' ('By rule, as here, or some other way?')
'How should phenomena of agreement and concord be handled?' ('How to guarantee that **the three...balloon* will not occur?')
'What constitutes a class? a subclass? (of adjectives, for example?)'
'What are the strengths and weaknesses of context-free phrase structure rules such as those in (10)?'

4. The Generation of Specific Noun Phrases Using Context-Sensitive Phrase Structure Rules

A programming language that provided only context-free phrase structure rules would be of limited usefulness to linguistics students. Clearly, there must be a better way to write a grammar than by simply enlarging the rule set in (10).

NLP uses rules that are basically context-free phrase structure rules, but with two important additions. Arbitrary CONDITIONS on the operation of a rule can be added in parentheses following the segment name on the left of the arrow; and arbitrary STRUCTURE-BUILDING ACTIONS can be stated in parentheses following each segment name on the right of the arrow. These additions result in 'augmented' phrase structure rules. The rule conditions make NLP equivalent to a context-sensitive grammar; and the actions give it the full power of transformational, or unrestricted rewriting, rules. So for example:

(11) NP(COLOR) --> ADJ(WORD=COLOR(NP)) NP(-COLOR)

Rule (11) says that a noun phrase which has a color connected with it (a condition) can be rewritten as an adjective followed by a new NP. The word associated with the adjective will be set equal to the color associated with the NP (an action); and the new NP will have the color removed from it (via another action), since that color will have been assigned to the premodifying adjective.

A rule like (11) suggests that segment types can have associated with them various kinds of information. In fact, such collections of information are central to the workings of NLP. The linguistic information which is manipulated by rules is carried in the form of ATTRIBUTES, each attribute having an assigned VALUE. (For instance, COLOR='RED' would mean that the attribute COLOR has been given the value RED.) A collection of attributes can be grouped together to form a RECORD. There are two types of records: NAMED RECORDS, which hold static and largely idiosyncratic information (these can be thought of as entries in a lexicon), and SEGMENT RECORDS, which hold dynamic information and are created and destroyed during the production of text. What we have been calling segment types, when they have attribute-value information associated with them, are called segment records.

Rather than being introduced by rule as in (10), lexical items can be stored as named records, with associated syntactic, phonological, and semantic feature information. Collections of information larger than single words can be grouped together among the named records too.

Suppose that the phrases *those three sheep* and *two big red balloons* were considered to be collections of information in the following manner:

```
(12) NAMED RECORDS
          RECORD1  (SUP='SHEEP',DET='THOSE',NUM='THREE')
          RECORD2  (SUP='BALLOONS',COL='RED',NUM='TWO',SIZ='BIG')
```

For simplification, the values of attributes in (12) are treated as words. They need not be. More primitive feature values can be used. The words themselves can be located in some other named record, and be retrieved by rule according to their feature makeup.

Attribute names in (12) are self-explanatory, except for SUP. SUP is short for SUPERSET, and should really be used to store some kind of hierarchical type information. However, for present purposes we are using it to store the name of the head of the phrase.

As a second programming exercise, try writing the NLP rules, with conditions and structure-building actions, that could operate on the records in (12) to produce two noun phrases, *those three sheep*, and *two big red balloons*. The idea here is to write a maximally general set of rules, that will handle not only the information in (12), but also all similar sets of information about entities that someone might want to express in English noun phrases. Remember that order counts: starting each time at the top of the list, the first encoding rule will be chosen whose left-hand side matches the information contained in the record being worked on. When your rules are ready, you can give a simple command to the computer:

```
ENCODE  NP  RECORD1  (or  ENCODE  NP  RECORD2).
```

This command will cause the program to take the information in RECORD1 or RECORD2, look for the first NP rule whose conditions are met by that information, apply the rule, and go on from there.

Two other NLP conventions are helpful here:

(a) The cent sign means to build record structure by copying a record from the left side of the arrow into a new record created on the right side: e.g., NP --> NOUN(¢NP) means to create a NOUN record and copy into it all information from the NP record on the left. If a segment type named on the right is the same as the one on the left, an automatic copy will be made and there is no need for the ¢ sign.

(b) If there is no rule to operate on a segment record, a piece of information from that record will be printed out. Present conventions specify that what gets printed in this case will be the value of the SUP attribute of that record.

The rules devised should look something like:

```
(13) a)   NP(DET)   -->  #  DET(SUP=DET(NP))   NP(-DET)
     b)   NP(NUM)   -->  #  NUM(SUP=NUM(NP))   NP(-NUM)
     c)   NP(SIZ)   -->  #  SIZ(SUP=SIZ(NP))   NP(-SIZ)
     d)   NP(COL)   -->  #  COL(SUP=COL(NP))   NP(-COL)
     e)   NP        -->  #  NOUN(¢NP)
```

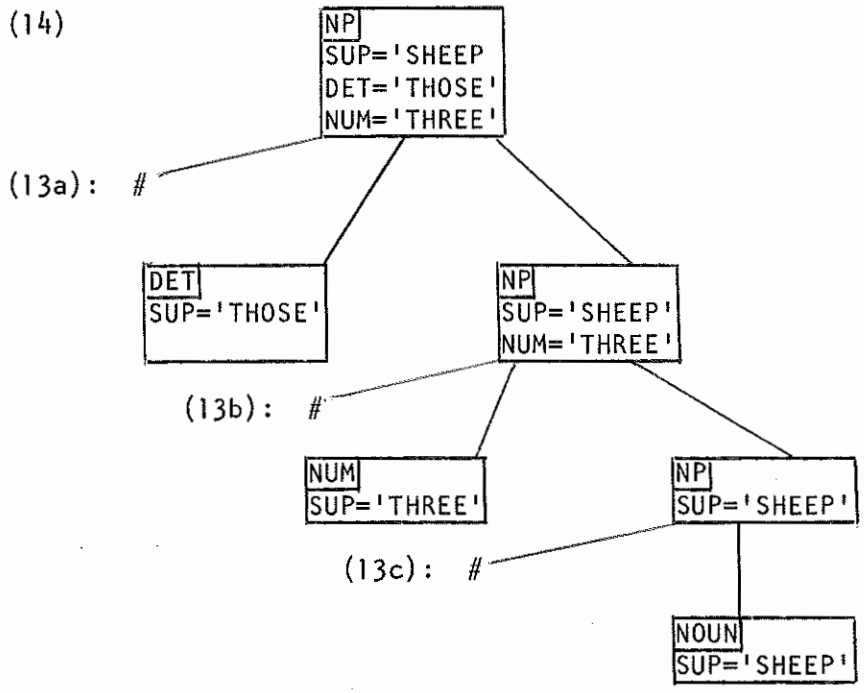
Rule (13a) says that a NP record that has a DET attribute gets rewritten as a DET record (whose SUP is set equal to the value of the DET attribute of the NP), followed by a new NP (which starts as an automatic copy of the NP on the left of the arrow, but is changed by having its DET attribute removed). If this subtraction is not done, the new record will qualify again and again for rule (13a), and you will have your first example of infinitely looping recursion. Compare the insertion of spaces ("#") here and in (10).

Moving down the list of rules, attributes of the record are converted, one by one, into lexical items: first the determiner (if called for), then a numeral, then a size attribute, then a color word, and finally the head noun, which starts as the SUP of the original NP record and is left after all other attributes have been removed. This ordering of the rules guarantees that adjectives will be put out in proper left-to-right order: *two big red balloons* and not **red big two balloons*. (13) is also maximally general in the sense that any collection of this sort of information about an entity can be given proper English syntax by these rules. All possible attributes need not be present in the record to start with; those rules whose conditions are met will apply, and the rest will not.

Given the records in (12) and the rules in (13), the command

ENCODE NP RECORD1

will produce the string *those three sheep* in the following way:



5. The Dative Movement Transformation

The transformational view of language, with its emphasis on precise notation and the detailed interaction of rules, lends itself beautifully to a computational approach. For each transformation, an NLP rule or rules can be written that will produce strings which are related in the desired manner. These rules might be thought of as programmed transformations. Using the computer, it is possible to experiment easily and decisively with rule interdependencies, ordering hypotheses, lexical restrictions, constraints, and other transformational problems. Of course, the bigger the problem, the bigger the rule set will have to be.

In Chapters 5 and 6, Akmajian and Heny discuss a formal rule that relates such English sentences as *Mary gave a book to the man,* and *Mary gave the man a book* (op cit.:183). The next programming task will be to write rules that would account for this Dative Movement phenomenon as discussed by Akmajian and Heny. We start, as before, with a collection of named records that hold some basic information. Again, for simplicity's sake, the information will be extremely sketchy and words will be used as the values of attributes. It should be fairly easy to see how this simplistic approach can be made more interesting via the use of word classes, morphological processing, feature notation, and more detailed syntactic deep structure data.

One minor modification has been made in these records: in NLP, it is possible to omit the SUP attribute label and have it understood, if a value is given in single quotes by itself. Where 'SHEEP' is printed, the program understands SUP='SHEEP', and so on.

```
(15) NAMED RECORDS
          RECORD1  ('SHEEP', DET='THOSE',NUM='THREE')
          RECORD2  ('JOHN')
          RECORD3  ('GAVE',DO='RECORD1',IO='RECORD2',DM='YES')
```

RECORD1 is our old friend from above, the record that becomes, via the rules in (13), *those three sheep*. RECORD2 has only one attribute, a SUP, and so will produce the terminal string *John* when processed by the NP rules. RECORD2 can be looked on as an entry for a lexical item. RECORD3 has a SUP which will ultimately appear as *gave*, a DO attribute that points to RECORD1, an IO attribute that points to RECORD2, and a DM attribute that is given a value of 'YES'. From the information presented in RECORD3, we would expect to generate a verb phrase with *gave* as its head. If Dative Movement is involved, the two related verb phrases,

gave those three sheep to John
gave John those three sheep

should be produced. What rules might be used?

```
(16) a)   VP(DO,IO,DM)   -->  VERB(¢VP,-DO,-IO,-DM)
                              NP(¢IO(VP))
                              NP(¢DO(VP))
     b)   VP(DO,IO)      -->  VERB(¢VP,-DO,-IO)
                              NP(¢DO(VP))
                              PP(¢IO(VP),PREP='TO')
```

If the command ENCODE VP RECORD3 is issued, rule (16a) will take RECORD3 and generate, from it, a verb which is a copy of the VP but with all information except the SUP removed; then a noun phrase which is a copy of the IO attribute of the verb phrase; and finally another NP which is a copy of the DO attribute: in effect, *gave John those three sheep*. (Of course, the rules in (13) will also be needed to create the NPs *those three sheep* and *John*.)

If we want to generate the related string without Dative Movement, we instruct the program to drop the DM attribute from RECORD3:

```
RECORD3  (-DM)
```

and then issue ENCODE VP RECORD3 again. This time rule (16a) will not be used, since its condition requires the presence of a DM attribute.

Instead, the program will fall through to rule (16b), which requires only the DO and IO attributes. Rule (16b) generates the verb followed by a noun phrase (*those three sheep*) and then a prepositional phrase, which is a copy of the IO attribute of the VP plus a new PREP attribute. An additional rule is needed to encode the PP:

(17) PP --> PREP(SUP=PREP(PP)) NP(¢PP,-PREP)

(13), (16b) and (17) will generate *gave those three sheep to John*.

Not accounted for in these elementary rules are several important facts about Dative Movement, e.g.:

only a certain class of verbs (of which *give* is representative) can take an indirect object;
if the direct object (DO) is a pronoun, Dative Movement (DM) must not apply.

Consider how facts such as these might be expressed in the rules of (16).

6. Incorporating Case into the Grammar

It is essential to realize that, so long as you abide by the stated conventions, the writing of NLP rules is a completely arbitrary process. Rules that have been presented here have followed a fairly traditional syntactic format. But it would be just as easy to write X-bar rules, or lexical-functional rules, or relational grammar rules, or semantic rules, or to test new ideas of constituent structure. To demonstrate this fact, we look briefly at how notions of Case Grammar can be incorporated into the rules we have worked with so far.

Verbs that undergo Dative Movement are verbs of transfer. These require an Agent (the one who makes the transfer), a Receiver, and an Object (the thing that gets transferred). The rules in (16) referred to Indirect Object (IO) and Direct Object (DO); but these attributes might as well have been called Receiver (RECV) and Object (OBJT), respectively. If we alter the records in (15) slightly,

```
(18) RECORD1  ('SHEEP',DET='THOSE',NUM='THREE')
     RECORD2  ('JOHN')
     RECORD3  ('GAVE',AGNT='RECORD4',OBJT='RECORD1',RECV='RECORD2')
     RECORD4  ('MARY')
```

change IO to RECV and DO to OBJT in (16), and add one more rule--this

time at sentence level,

(19) SENT(AGNT) --> NP(¢AGNT(SENT)) VP(%SENT,-AGNT)

our program will be able to generate whole sentences of the form, *Mary gave those three sheep to John; Mary gave John those three sheep.* By further modifying the content of the records, a large number of interesting sentences can be produced. Most of these sentences will probably fall into the 'transfer' class; but note that the rules developed here will also generate simple intransitive sentences (*Mary ran; John followed*). Remember that if there is no rule to operate on a record, the value of the SUP attribute will be printed out anyway. And an interesting modification of the VP rules would allow for the production of simple transitives, like *John followed the sheep.*

As a final programming task, try to augment and modify the rules in (16) so as to produce, in an efficient way, a transitive verb phrase.

7. Summary

The careful and precise study of the structure of language--a study which is central to the nature of linguistics--can be both enabled and encouraged by adding a computational course to the linguistics curriculum. For maximum benefit, this course should offer a programming language that is congenial to linguists. NLP is such a language. Students coming from different academic backgrounds have enjoyed the course at Hofstra, which strives to make computational techniques more familiar to linguistics students, and to make the nature of human language more understandable to all.

8. Appendices

Following are two appendices. The first is a collection of NLP records and rules developed in this paper. The rules have been modified to be consistent with each other, and some additions have been made to suggest possible solutions to the problems posed at the end of Sections 5 and 6. (The solutions offered here are by no means the only ones that would work.) These rules form a very simple beginning for a computational grammar. The second appendix presents a short outline for a course, like the one discussed here, which serves to introduce both English syntax and computational linguistics.

APPENDIX 1

```
NAMED RECORDS
     RECORD1  ('SHEEP',DET='THOSE',NUM='THREE')
     RECORD2  ('JOHN')
     RECORD3  ('GAVE',AGNT='RECORD4',OBJT='RECORD1',RECV='RECORD2',DM)
     RECORD4  ('MARY')

ENCODING RULES
     (1)  SENT(AGNT)  -->  NP(¢AGNT(SENT))  VP(%SENT,-AGNT)
     (2)  VP(OBJT,RECV,DM,'TRNSFR'.ISIN.VCLASS($(SEG)),HEAD(OBJT).
                                                          NE.'PRON')
                      -->  VP(-OBJT,-RECV,-DM)
                           NP(¢RECV(VP))
                           NP(¢OBJT(VP))
     (3)  VP(RECV)    -->  VP(-RECV)
                           PP(¢RECV(VP),PREP='TO')
     (4)  VP(OBJT)    -->  VP(-OBJT)
                           NP(¢OBJT(VP))
     (5)  VP          -->  #  VERB(¢VP)
     (6)  PP          -->  #  PREP(SUP=PREP(PP))
                           NP(¢PP,-PREP)
     (7)  NP(DET)     -->  #  DET(SUP=DET(NP))  NP(-DET)
     (8)  NP(NUM)     -->  #  NUM(SUP=NUM(NP))  NP(-NUM)
     (9)  NP(SIZ)     -->  #  SIZ(SUP=SIZ(NP))  NP(-SIZ)
     (10) NP(COL)     -->  #  COL(SUP=COL(NP))  NP(-COL)
     (11) NP          -->  #  NOUN(¢NP)
```

WEEK	TOPIC	ASSIGNMENTS
1	Introduction: summarize natural language processing to date.	Read handouts.
2	Review of traditional grammar: parts of speech; constituent constructions; levels of grammatical analysis.	Handouts.
3	Rules in general: regular rules and phrase structure context-free rules. Introduce NLP encoding (APSG).	Handouts. Akmajian & Heny, Ch.2.
4	Regular and PSCF rules, continued. Basic English sentence structure. Random generation of NPs using NLP.	Handouts. Akmajian & Heny, Ch.3.
5	Context-sensitive rules. Affix Hop. Aspect and modality. Random generation of intransitive VPs.	Handouts.
6	Transformational rules: justification, what they can do. Passive, Q, Neg, Tag, DO-auxiliary. Random generation of Ss.	Handouts. Akmajian & Heny, Ch.4.
7	T-rules continued: reflexivization, imperative, THERE-insertion, dative, particle shift. Ordered generation of NPs.	Handouts. Akmajian & Heny, Ch.5.
8	MID-TERM EXAM.	Handouts.
9	Semantics: constraints on deletion; logical grammar; problems of scope.	Handouts. Akmajian & Heny, Ch.7.
10	Case grammar: embedding of propositions; extraposition; Equi; complementizers.	Handouts. Akmajian & Heny, Ch.8.
11	The lexicon: word classes; computer poetry; deverbal forms (*-ing, -ed* words).	Handouts. Akmajian & Heny, Ch.9.
12	Paraphrase and ambiguity. Zero derivation.	Handouts.
13	Generating languages other than English.	Handouts.
14	Summarize; prepare for final; prepare for future.	Work work work enjoy.

REFERENCES

Akmajian, Adrian and Frank Heny. 1975. *An Introduction to the Principles of Transformational Syntax*. Cambridge, Massachusetts: MIT Press.

Chomsky, Noam. 1957. *Syntactic Structures*. The Hague: Mouton & Co.

Heidorn, G.E. 1972. Natural Language Inputs to a Simulation Programming System. *Technical Report NPS-55HD72101A*. Monterey: Naval Postgraduate School.

_____. 1975. Augmented Phrase Structure Grammars. In B.L. Nash-Webber and R.C. Schank (eds.), *Theoretical Issues in Natural Language Processing*. Association for Computational Linguistics.