Pronouns

Kelly Roach

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-9

Pronouns

Thesis by

Kelly Roach

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

California Institute of Technology

Pasadena, California

1980

(Submitted May 22, 1980)

Caltech-CS-TR-88-9

Table of Contents

Chapter 0. Introduction.

Certain substitutions and abbreviations occur in English which are not well understood yet that we would like to understand better so that we may implement them in computer natural language systems intended for man-machine communication. These include pronouns and other function words like those below in Figure 0.1 acting both in isolation and with each other.

he	him	his	himself
she	her	hers	herself
it	its	itself	they
them	their	theirs	themselves
this	that	these	those
one	ones	oneself	other
others	all	none	some
any	each	which	what
do	does	did	so
another			

Figure 0.1. Pronouns And Other Function Words.

As well, we have noun phrases modified by demonstratives, head deletion, and Equi-NP Deletion. Bloomfield [2] defined substitution as a replacement operation.

> A substitute is a linguistic form or grammatical feature which, under certain conventional circumstances, replaces any one of a class of linguistic forms. Thus, in English, the substitute I replaces any singular-number substantive expression, provided that this substantive expression denotes the speaker of the utterance in which the substitute is used.

In this paper we will be concerned with pronouns. Possibly because this will be the only chance we get, we should note the wide variety of substitution mechanisms in general. Examples (0.2)-(0.11) are from Sag [27].

(0.2) Do It Anaphora
Jerry won't prove that theorem; Alice will do it.
[do it=prove that theorem]

(0.3) Sentential It Anaphora
I believe that she means business and you'd better believe it too.
[it=that she means business]

(0.4) Null Complement Anaphora
They asked me to leave but I refused PHI.
[PHI=to leave]

(0.5) Ones Pronominalization
Betsy has a blue car, and Randy has a red one.
[one=car]

(0.6) Verb Phrase Deletion
Joan wouldn't eat a Quarter Pounder, but Annie would PHI.
[PHI=eat a Quarter Pounder]

(0.7) Sluicing
Someone has drunk my entire six-pack of Schlitz Light, but I don't know who PHI.
[PHI=has drunk my entire six-pack of Schlitz-Light]

(0.8) Stripping
Gwendolyn snorts cocaine, but PHI1 not PHI2 in her own apartment.
[PHI1=Gwendolyn (does), PHI2=snort cocaine]

(0.9) Gapping
Erichman duped Haldeman and Nixon PHI Mitchell.
[PHI=duped]

(0.10) Conjunction Reduction
Mitchell lied to the committee and PHI was sentenced last year.
[PHI=Mitchell]

(0.11) So Anaphora
Mitchell said he was innocent and Nixon said so too.
[so=he was innocent]

To this list we can add pronominalizations. Examples (0.12)-(0.14) are from Lees and Klima [22].

(0.12) Reflexive Pronominalization
Mary's father supported himself.
[himself=Mary's father]

(0.13) Pronominaliztion
Mary's father supported her.
[her=Mary]

(0.14) Reciprocal Pronominalization
John and Mary kissed each other.
[each other=John and Mary]

And we might add (0.15) and (0.16) as well.

(0.15) Head Deletion
Joan's cat purrs but Mary's PHI doesn't.
[PHI=cat]

(0.16) Equi-NP Deletion
John is afraid of PHI cutting himself.
[PHI=John's]

Clearly, this list starts to grow very large with addition or refinement and it is probably safe to say that many volumes could be written on substitution processes without putting it to bed. This paper is about pronouns and chaining of pronouns, and so is much narrower in scope. But this is rot much comfort if the goals are not clearly in

sight. We are just as lost in the middle of Lake Michigan as we are in the middle of the Pacific Ocean if we don't have a horizon to steer us by.

Part of the problem with investigations of anaphora today is that there is no horizon to steer by. Even though work on anaphora continues in an intelligent way, little progress is being made towards a really comprehensive theory. Instead we have a lot of scattered and indepedent results.

One goal of this paper, besides talking about pronouns, is to seek out an algorithmic framework on which to build a theory. Accordingly, various data structures such as nodes, C-S-N trees, and chaining tables are created for this purpose. Hopefully, the reader will recognize these data structures as too simplistic and will be moved to improve upon them. This paper is by no means at all a solution to pronouns. At best it may be a small compass in the middle of Lake Michigan. But this is our approach.

Chapter 1. Fundamentals.

Introduction.

This chapter describes notation and basic ideas that will be used throughout this paper. Hopefully, most of the notation described in this chapter is already familiar to the reader, but if not, then this chapter should be self-contained enough to be understandable by a reader with less experience.

Sentences.

Sentences are numbered and are kept separate from the text of discussion for ease of reference. For example, (1.1) is from Huddleston [13] and is an example of a Bach Peters sentence.

(1.1) The boy who was fooling her kissed the girl who loved him.

Ungrammatical sentences are prefixed with an asterisk and sentences of questionable grammaticality are prefixed with a question mark. Here (1.3) is from Chomsky [5].

(1.2) *John killed herself.
(1.3) ?Colorless green ideas sleep furiously.

Subscripts are used to indicate identity between constitutents, meaning roughly that they mean the same thing or denote the same referent. More properly, we may think of constituents having the same subscript as being chained

together. Below, (1.4) and (1.5) are from Bresnan [3].

(1.4) Some studentsI think theyI are smarter than theyI are.
(1.5) *Some studentsI think some studentsI are smarter than some studentsI are.

Sometimes we enclose information in brackets at the beginning or end of a sentence. This same notation is also sometimes used as an alternative to subscripts in identifying constituents. Here, (1.6) is from Bresnan [3], (1.7) and (1.8) are from Roberts [25] and (1.9) is from Bloom and Hayes [1].

(1.6) My uncle has never ridden a camel, but his brother has, although it was lame. [it=camel]
(1.7) Men are mortal. [All men are mortal]
(1.8) Men are waiting. [Some men are waiting]
(1.9) [Seeing a picture of John Smith] That's John Smith.

A deletion site is indicated by a PHI. Example (1.10) is from Hockett [12].

(1.10) I like the fresh candy better than the stale PHI. [PHI=candy]

Deletion sites arising from transformations like Equi-NP Deletion are treated similar to pronouns in this paper. Although there are many different kinds of deletion sites with distinct properties, we won't pay attention to this distinction in this paper.

The symbol = is used between sentences to indicate that they are equivalent, while the symbol <> is used between sentences to indicate that they are not equivalent. Below, (1.11)-(1.14) are from Ross [26].

(1.11) If John can, he will do it. =
(1.12) If he can, John will do it.
(1.13) John will do it if he can. <>
(1.14) He will do it if John can.

Noun Phrases.

Quantified noun phrases are noun phrases modified by quantifiers. Examples (1.15)-(1.18) are quantified noun phrases.

(1.15) all female astronauts
(1.16) at least 10 sexual perverts
(1.17) many notorious criminals
(1.18) nearly a dozen Unicorns

Genitives are possessive noun phrases. Examples (1.19)-(1.22) are genitives.

(1.19) Uncle Iggy's
(1.20) my cobra's
(1.21) the Nazi war criminal's
(1.22) the alien creatures'

A noun phrase can be generic, specific, or nonspecific, indicated respectively by (1.23)-(1.25) from Kuno [16].

(1.23) A cat is a malicious animal. [generic]
(1.24) I have a cat at home, but I hate it. [specific]
(1.25) I want to get a cat for myself. [nonspecific]

A plural noun phrase can be collective or distributive. Examples (1.26)-(1.28) are from Fauconnier [7].

(1.26) The men gathered. [collective]
(1.27) The men took off their hats. [distributive]
(1.29) The men carried the couch. [ambiguous]

Sentence (1.29) is ambiguous because it can mean either (1.30) or (1.31).

(1.30) Each man of the men carried the couch.
(1.31) The team of men carried the couch.

Smith [30] has also noticed this distinction. This explains why (1.32)-(1.35) below are ambiguous.

(1.32) John and Mary bought the new book by John Steinbeck.
(1.33) Bricks and stones make strong walls.
(1.34) George and Marmaduke have dogs.
(1.35) Gerry likes ice cream and cake.

Pronouns.

Pronouns are cross-classified by person, plural, gender, animate, reflexive, attributive possessive, and predicative possessive features among others.

First person pronouns are given in (1.36).

(1.36) I, me, myself, my, mine

Second person pronouns are given in (1.37).

(1.37) you, yourself, yourselves, your, yours

Third person pronouns are given in (1.38).

(1.38) she, he, it, they, her, him, them, herself, himself, itself, themselves, his, its, their, hers, theirs

Singular pronouns are given in (1.39).

(1.39) I, me, myself, my, mine, you, yourself, your, yours, she, he, it, her, him, herself, himself, itself, his, its, hers

Plural pronouns are given in (1.40).

(1.40) you, yourselves, your, yours, they, them, themselves, their, theirs

Pronouns with female gender are given in (1.41).

(1.41) she, her, herself, hers

Pronouns with male gender are given in (1.42).

(1.42) he, him, himself, his

Animate pronouns are given in (1.43).

(1.43) I, me, myself, mine, you, yourself, youselves, your, yours, she, he, they, her, him, herself, himself, themselves, his, their, hers, theirs

Inanimate pronouns are given in (1.44).

(1.44) it, they, them, itself, themselves, its, theirs

Reflexive pronouns are given in (1.45).

(1.45) myself, yourself, yourselves, herself, himself, itself, themselves

Attributive possessive pronouns are given in (1.46).

(1.46) my, your, her, his, its, their

Predicative possessive pronouns are given in (1.47).

(1.47) mine, yours, hers, his, its, theirs

Besides the pronouns given above, we also have ones pronouns and reciprocal pronouns. Ones pronouns are given in (1.48).

(1.48) one, oneself, one's

Reciprocal pronouns are given in (1.49).

(1.49) each other, one another, each other's, one another's

Features.

We use three kinds of features in this paper. The

symbol + indicates presence of a feature. The symbol - indicates absence of a feature. And the symbol ? indicates that the presence or absence of a feature is either unspecified or not applicable. In the coming chapters, we will speak of agreement of features. A ? feature agrees with any other feature. The only time two features do not agree is when we are comparing a + and a - feature. Using = to indicate agreement and <> to indicate nonagreement, we have Figure 1.50.

+ = +	+ = ?	+ <> -
? = +	? = ?	? = -
- <> +	- = ?	- = -

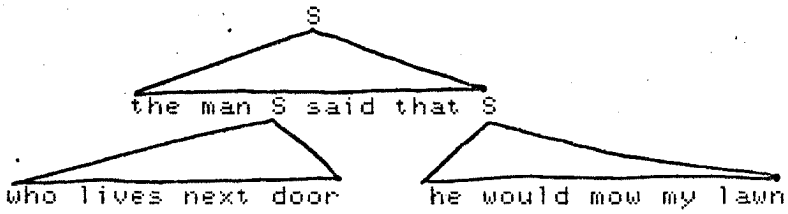
Figure 1.50. Agreement And Nonagreement Between Features.

Parse Trees.

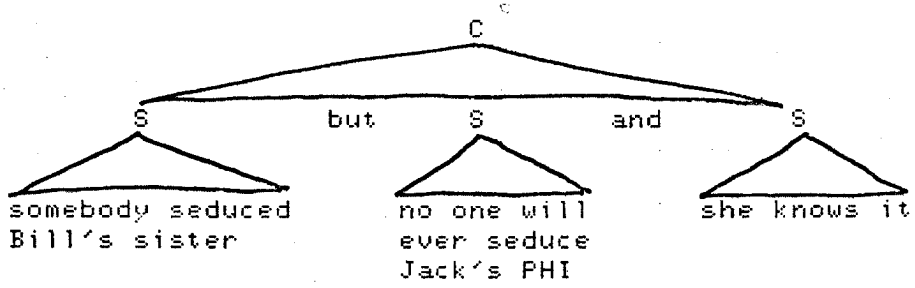
Sentence parse trees are only drawn schematically in this paper as extra detail is unnecessary. Parse trees shown more or less represent the surface structure of a sentence. Clause dominating nodes are labelled S and clause conjoining nodes are labelled C. In this paper, genitives and adjectives are not treated as arising from transfomations, but as occuring in the base component. Below, example (1.51) is from Huddleston [13] and example

(1.52) is from Grosu [9].

(1.51) The man who lives next door said that he would mow my lawn.



(1.52) Somebody seduced Bill's sister, but no one will ever seduce Jack's and she knows it.



Clauses.

Adverbial clauses are clauses beginning with an adverb. Some examples are (1.53)-(1.57) below.

(1.53) after Fido made a mess on the carpet
(1.54) before George kisses Betty
(1.55) since John is an asshole
(1.56) until Cathy behaves herself
(1.57) although Lile flunked all his classes

Clauses complementized with that are that clauses. Example (1.58) is a that clause.

(1.58) that Snoopy is a cat

Clauses modified by the for-to transformation are

infinitive clauses. Example (1.59) is an infinitive clause.

(1.59) for Ruth to choose

Clauses modified by the possessive-ing transformation are genitive clauses. Example (1.60) is a genitive clause.

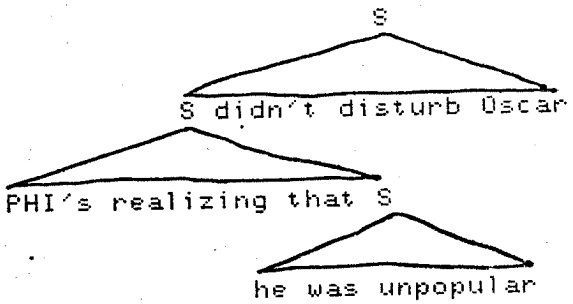
(1.60) Mary's kissing Bob

Clauses modified by WH-Fronting but not the Question Transformation and which modify noun phrases are relative clauses. Examples (1.61)-(1.65) are relative clauses.

(1.61) who ate five hamburgers
(1.62) that has a leaky faucet
(1.63) which doesn't run
(1.64) whom he gave it to
(1.65) whose life isn't worth a postage stamp

Clauses without embedded subordinate clauses are simplexes. In example (1.66) from Ross[26], the simplexs are (1.67)-(1.69). In example (1.70), from Huddleston [13], the simplexs are (1.71)-(1.73). In example (1.74), from Huddleston, the simplexs are (1.75)-(1.77).

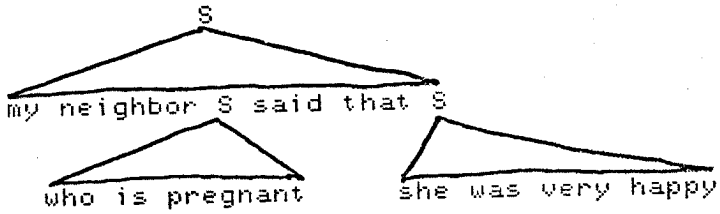
(1.66) Realizing that he was unpopular didn't disturb Oscar.



PHI precedes he
PHI commands he
PHI precedes Oscar
Oscar commands PHI
Oscar commands he

(1.67) S didn't disturb Oscar
(1.68) PHI's realizing that S
(1.69) he was unpopular

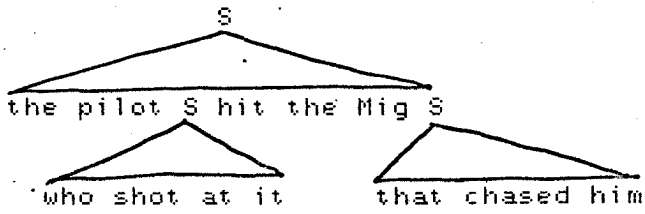
(1.70) My neighbor who is pregnant said that she was very happy.



neighbor precedes she
neighbor commands she

(1.71) my neighbor said that S
(1.72) who is pregnant
(1.73) she was very happy

(1.74) The pilot who shot at it hit the Mig that chased him.



the pilot precedes him — it precedes the Mig
the pilot commands him — the Mig commands it

(1.75) the pilot hit the Mig
(1.76) who shot at it
(1.77) that chased him

Precede and Command.

The precede and command relations, first described by Langacker [19], are defined below in (1.78) and (1.79).

(1.78) Precede Relation
A node A precedes another node B if
(a) neither A nor B dominates the other, and
(b) A occurs before B (in an inorder traversal)

(1.79) Command Relation
A node A commands another node B if
(a) neither A nor B dominates the other, and
(b) the S-node that most immediately dominates A also dominates B

Another relation that will be useful is the separate relation defined below in (1.80).

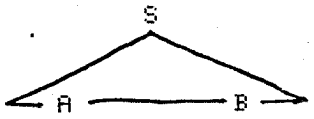
(1.80) Separate Relation
A node A is separate from another node B if
(a) neither A nor B dominates the other, and
(b) the lowest node in the tree dominating A and B is a C-node.

We will see that precede, command, and separate are useful in determining when pronominalization is or isn't

possible.

In example (1.81), A precedes B, A commands B, and B commands A. We don't have A precedes A, B precedes A, B precedes B, A commands A, or B commands B.

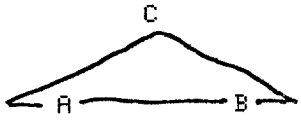
(1.81)



A precedes B
A commands B
B commands A

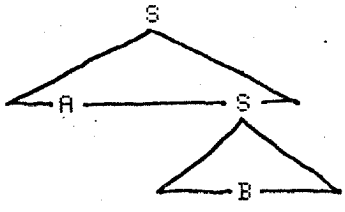
In (1.82), A precedes B, A is separate from B, and B is separate from A. In (1.83), A precedes B and A commands B. In (1.84), A precedes B and B commands A. In (1.85), A precedes B, A is separate from B, and B is separate from A.

(1.82)



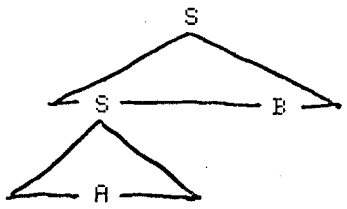
A precedes B
A is separate from B
A precedes B

(1.83)



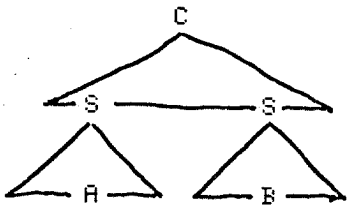
A precedes B
A commands B

:.84)



A precedes B
B commands A

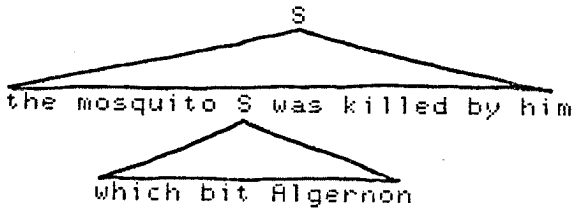
:.85)



A precedes B
A is separate from B
B is separate from A

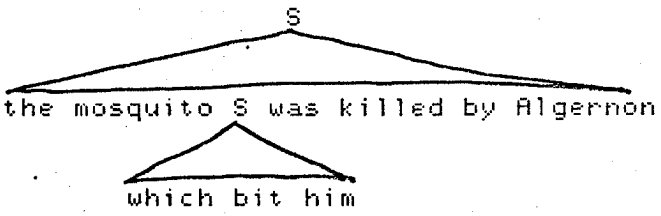
:amples (1.86)-(1.89) are from Langacker.

:.86) The mosquito which bit Algernon was killed by him. [him=Algernon]



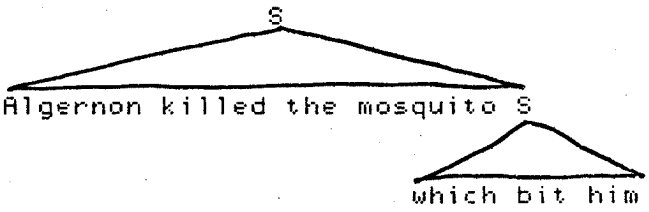
Algernon precedes him
him precedes Algernon

(1.87) The mosquito which bit him was killed by Algernon. [him=Algernon]



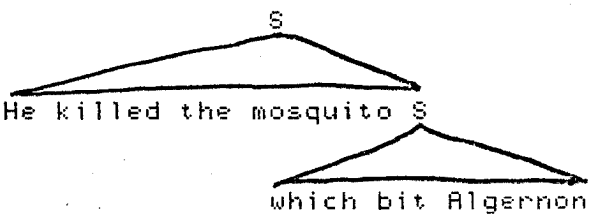
him precedes Algernon
Algernon precedes him

(1.88) Algernon killed the mosquito which bit him. [him=Algernon]



Algernon precedes him
him commands Algernon

(1.89) Algernon killed the mosquito which bit him. [him=Algernon]



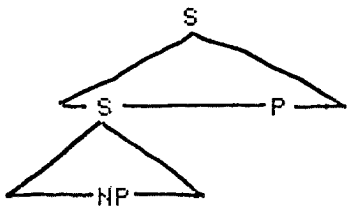
he precedes Algernon
Algernon commands he

The precede and command rule, essentially as stated by Langacker, is given in (1.90) below.

(1.90) Precede and Command Rule
A pronoun P may be used to pronominalize a noun phrase NP unless
(a) P precedes NP, and
(b) P commands NP or P is separate from NP

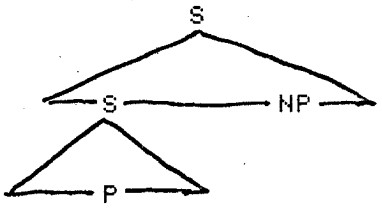
Note that the precede and command rule explains the grammaticality and ungrammaticality of (1.86)-(1.89). These further examples from Ross [26] should drive the point home.

(1.91) After John Adams woke up, he was hungry. [he=John Adams]
(1.92) That Oscar was unpopular didn't disturb him. [him=Oscar]
(1.93) For your brother to refuse to pay taxes would get him into trouble. [him=your brother]
(1.94) Anna's complaining about Peter infuriated him. [him=Peter]
(1.95) The possibility that Fred will be unpopular doesn't bother him. [him=Fred]



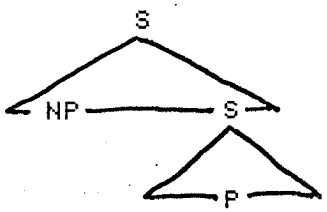
NP precedes P
P commands NP

(1.96) After he woke up, John Adams was hungry. [he=John Adams]
(1.97) That he was unpopular didn't disturb Oscar. [he=Oscar]
(1.98) For him to refuse to pay taxes would get your brother into trouble. [him=your brother]
(1.99) Anna's complaining about him infuriated Peter. [him=Peter]
(1.100) The possibility that he will be unpopular doesn't bother Fred. [him=Fred]



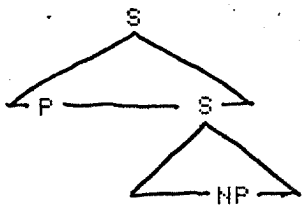
P precedes NP
NP commands P

(1.101) John Adams was hungry after he woke up. [he=John Adams]
(1.102) Oscar wasn't disturbed that he was unpopular. [he=Oscar]
(1.103) It would get your brother into trouble for him to refuse to pay taxes. [him=your brother]
(1.104) Peter was infuriated at Anna's complaining about him. [him=Peter]
(1.105) Fred isn't bothered by the possibility that he will be unpopular. [he=Fred]



NP precedes P
NP commands P

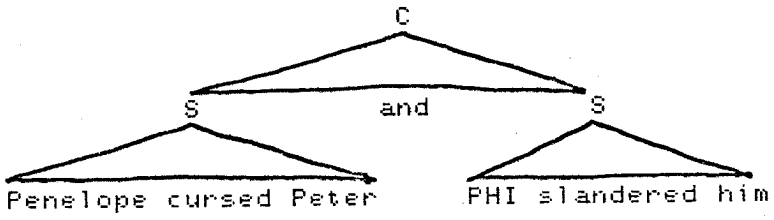
(1.106) *He was hungry after John Adams woke up. [he=John Adams]
(1.107) *He wasn't disturbed that Oscar was unpopular. [he=Oscar]
(1.108) *It would get him into trouble for your brother to refuse to pay taxes. [him=your brother]
(1.109) *He was infuriated at Anna's complaining about Peter. [he=Peter]
(1.110) *He isn't bothered by the possibility that Fred will be unpopular. [he=Fred]



P precedes NP
P commands NP

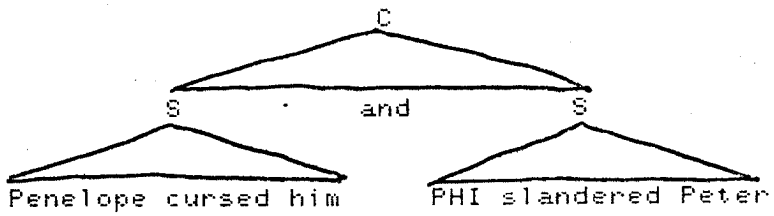
Examples (1.111) and (1.112) from Langacker illustrate the precede and command rule for conjoined structures.

(1.111) Penelope cursed Peter and slandered him. [him=Peter]



Peter precedes him
Peter is separate from him
him is separate from Peter

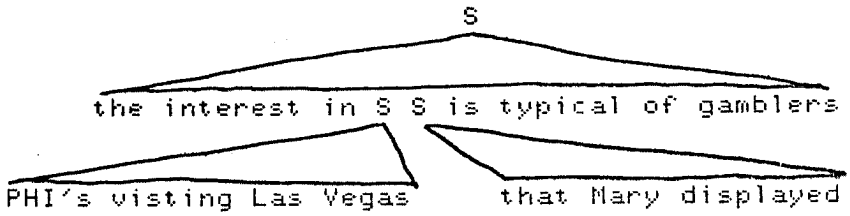
(1.112) *Penelope cursed him and slandered Peter.
[him=Peter]



him precedes Peter
him is separate from Peter
Peter is separate from him

Examples (1.113) and (1.114) adapted from Chiba [3] involve Equi-NP Deletion.

(1.113) The interest in visiting Las Vegas that Mary displayed is typical of gamblers.



PHI precedes Mary

Chapter 2. The Resolution Module.

Introduction.

In the previous chapter we touched upon some basic notions such as the precede, command, and separate relations. We will see in the coming chapters how these concepts give rise to a very promising approach to the problem of pronoun resolution.

The algorithm we shall describe won't be complete in the sense that we will elaborate and refine it in later chapters and after we are done it will need elaboration and refinement, but it will be set in firm soil so that we have a foundation on which to build. Because personal and reflexive pronouns are easiest, these are the pronouns we shall consider first. But before we go any farther, let us take time out to indicate something of the environment and structure of the module that does resolving of pronouns in a natural language system, the Resolution module.

The Environment.

The center of a natural language system is the Language Processor module which is divided into five submodules. These are the Language Driver, Preprocesor, Parser, Semantic Processor, and Output Processor as indicated in Figure 2.1.

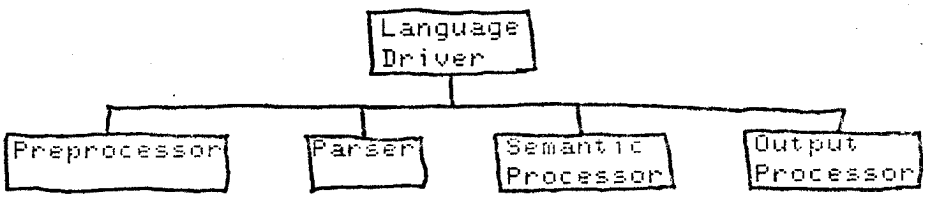


Figure 2.1. Submodules Of The Language Processor.

Briefly, from the point of view of the Language Processor, the following happens. A user types input at a terminal which is picked up by the Operating System of the natural language system. The Operating System maintains information about the user including the language version he is in as well as his state in that version. The user's state is known as his prefix. The Operating System, after picking up a user's input calls a Process Input routine of the Language Driver in the Language Processor. Once in the Language Driver, the first module to be called upon is the Preprocessor.

The Preprocessor in the Language Processor compresses blanks in the input string, straps right and left delimiters about it, recognizes and builds parsing graph arcs over identifiers and numbers, and looks the identifiers up in the lexicon. After calling the Preprocessor, the Language Driver calls the Parser.

The Parser in the Language Processor parses the output of the Preprocessor using an algorithm such as the Kay algorithm and can handle any general rewrite rule grammar.

Of course, since a sentence may be ambiguous, more than one system parse tree may be passed back by the Parser. If no good parsings are found, then the Syntax Diagnostics routine of the Syntax Diagnostics module of the natural language system is called. Otherwise, if there are good parsings, then the Language Driver calls the semantic Processor on the output of the Parser.

The Semantic Processor is driven by the syntax of a system parse tree into making calls on semantic routines which can be postprocedures (called on their arguments after their arguments evaluate themselves), preprocedures (called on their arguments before their arguments evaluate themselves), and syntax procedures (called at syntax time during parsing before preprocedures and postprocedures are called during semantic processing). On return to the Language Driver, the Language Driver calls the Output Processor on the output of the Semantic Processor.

The Output Processor does some relatively menial processing such as removing duplicate lines from the output line list which will be sent back to the Operating System. The Output Processor is able to handle ambiguous output and removes diagnostic messages if at least one of the outputs is good.

On completion of the call on the Output Processor, the Language Driver returns to the Operating System and the Operating System displays the output line list on the user's

terminal, at the same time updating its information on the user.

From the discussion of the precede, command, and separate relations in the previous chapter, we know that information about the syntax of the input sentence is critical to the resolving of pronouns in the input sentence. On the other hand, for semantic processing to carry out the processing it needs to carry out, the placing of information on the chaining of pronouns must already be placed in the system parse tree of the input sentence.

The logical conclusion of these two observations indicates that pronoun resolution takes place after parsing, but before semantic processing. This relationship of the Resolution module with the other modules of the Language Processor is indicated in Figure 2.2.

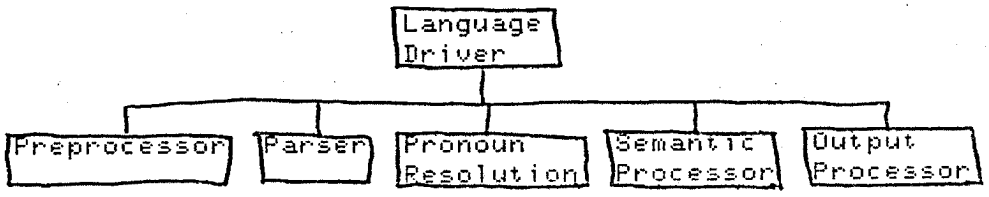


Figure 2.2. The Resolution Module Within The Language Processor.

In practice, this formulation may not be quite correct because there can be other versions than English which will have nothing to do with the Pronoun Resolution module and so

what we end up doing is making the Resolution module accessable via a semantic preprocedure which is associated with the parsing of the right delimiter of a sentence. So instead, what happens is that the first semantic preprocedure to be called will be the procedure which handles pronoun resolution.

Structure Inside The Resolution Module.

The Resolution module is partitioned into seven submodules besides a Global Declarations module. These are the Node Processor, Parser, Primary Utilities, Secondary Utilities, Table Processor, Table Interpreter, and resolution Driver modules. The reader should not confuse the Parser of the Language Processor with the Parser of the Pronoun Resolution module which have entirely different functions. The relationship of these submodules of the Resolution module is indicated below in Figure 2.3.

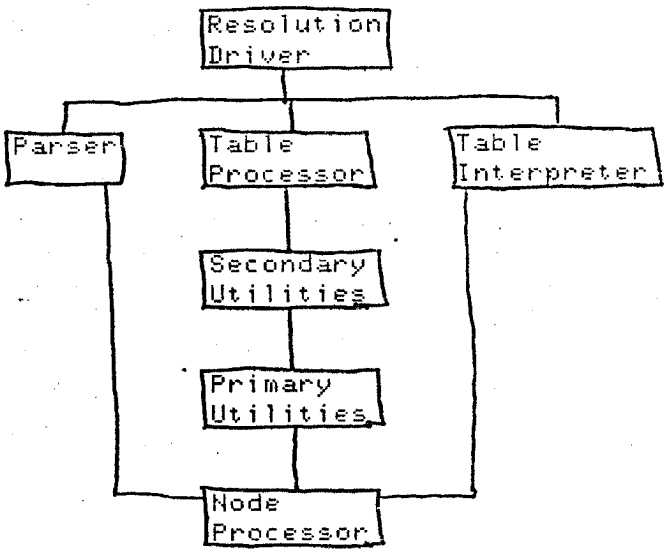


Figure 2.3. Structure Of The Resolution Module

Not shown is the Global Declarations module which does not have any procedures itself, but merely defines data structures. The Global Declarations submodule is accessible by all other submodules of the Resolution module.

Chapter 3. Global Declarations.

The global declarations module defines the data structures accessible to other modules within the pronoun resolution module. The global declarations module is shown below in Figure 3.1.

```
unit globals;       (Global Declarations Module)
    define
    const pnf=1;    (Pronoun Feature)
          fpf=2;    (First Person Feature)
          spf=3;    (Second Person Feature)
          tpf=4;    (Third Person Feature)
          plf=5;    (Plural Feature)
          gnf=6;    (Gender Feature)
          anf=7;    (Animate Feature)
          rpf=8;    (Reflexive Feature)
          nfeatures=8;      (Number of Features)

    type nodeid=(cnode,snode,nnode,enode); (Kinds of
                                            Nodes)
         feature=(plus, minus, question);  (Kinds of
                                            Features)
         features=packed array[1..nfeatures] of
                                             feature;
         string_pointer=^string;
         node_pointer=^node;
         node=record
                  number:integer;
                  uplink,downlink,leftlink,rightlink,
                  threadlink,nplink,chainlink,
                  collink:node_pointer;
                  ftr:features;
                  case id:nodeid of
                      cnode,snode:();
                      nnode:(lit:string_pointer;
                             endcollink,predlink,
                             succlink:node_pointer);
                      enode:(sub:char);
              end(node);

(       Node Fields According to Kind of Node

 (C,up,down,left,right,thread,number)
 (S,up,down,left,right,thread,number)
 (N,lit,ftr,up,down,left,right,thread,np,chain,col,
  endcol,pred,succ,number)
 (E,sub,ftr,np,chain,col)
)

implement
begin
end(globals);
```

Figure 3.1. The Global Declarations Module.

Basically, our data structures are C-S-N trees, chaining tables, and the nodes they involve. It will help to get some feel for these data structures before we go on to other chapters.

Nodes.

There are four kinds of nodes: C-nodes, S-nodes, N-nodes, and E-nodes. C-nodes, S-nodes, and N-nodes occur in C-S-N trees and correspond to conjoined structures, sentences, and noun phrases. E-nodes occur in chaining tables. The fields of the C-nodes, S-nodes, N-nodes, and E-nodes are as indicated in Figure 3.1.

C-S-N Trees.

A C-S-N tree has three kinds of nodes: C-nodes, S-nodes, and N-nodes. Link fields which are relevant to C-S-N trees are uplink, downlink, leftlink, rightlink, threadlink, predlink, and succlink. An example of a C-S-N tree is given in Figure 3.2.

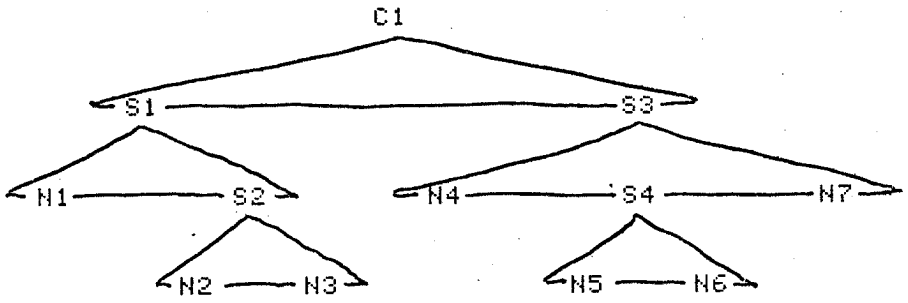


Figure 3.2. An Example Of A C-S-N Tree.

Chaining Tables.

A chaining table contains N-nodes, E-nodes, and one S-node for keeping track of the chaining table. Link fields relevant to chaining tables are nplink, chainlink, collink, endcollink, predlink, and succlink. Chaining tables and C-S-N trees are connected through their N-nodes. An example of a chaining table is given in Figure 3.3.

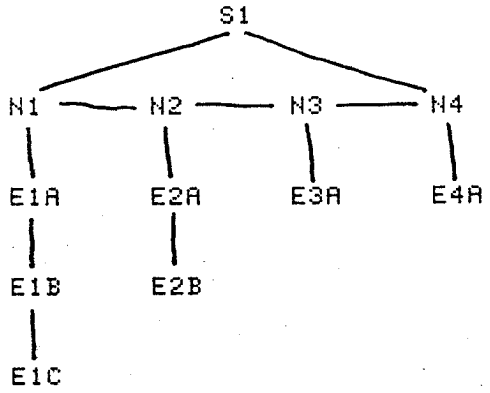


Figure 3.3. An Example Of A Chaining Table.

C-Nodes.

A C-node has the following fields: uplink, downlink, leftlink, rightlink, threadlink, and number. C-nodes correspond to conjoined sentences and conjoined subordinate clauses.

S-Nodes.

An S-node has exactly the same fields as a C-node and is only distinguished from a C-node by its nodeid. S-nodes

correspord to sentences and subordinate clauses.

N-Ncdes.

An N-node has the following fields: lit, ftr, uplink, downlink, threadlink, nplink, chainlink, collink, endcollirk, predlink, succlink, and number. N-nodes correspord to noun phrases without attached subordinate clause mcdifiers.

E-Ncdes.

An E-node has the following fields: sub, ftr, nplink, chainlink, and collink. An E-node may be thought of as a copy of its nplink with a slightly more defined set of features.

Lit Field.

The lit field of an N-node is a string pointer to the string that the N-node represents. The lit field is actually unnecessary in an N-node, but is convenient for displayirg intermediate results. Procedure listnode of the node prccessor and some other procedures that display intermediate results use this field.

Sub Field.

The sub field of an E-node is a character representing the subscript of the E-node. The sub field of an E-node, like the lit field of an N-node, is an unnecessary field, but is ccnvenient for displaying intermediate results.

Ftr Field.

The ftr field of an N-node or E-node is an array of features representing the feature set of the N-node or E-node to which it corresponds. A feature can be a plus, minus, or question as described in the previous chapter. The offsets pnf, fpf, spf, tpf, plf, gnf, anf, and rpf are used to access elements of the ftr array. The accessed elements are pronoun feature, first person feature, second person feature, third person feature, plural feature, gender feature, animate feature, and reflexive feature. The number of features is nfeature. Figure 3.4 shows some examples of the settings of features for some typical noun phrases.

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
John	-	-	-	+	-	-	+	-
flowers	-	-	-	+	+	?	-	-
he	+	-	-	+	-	-	+	-
them	+	-	-	+	+	?	?	-
I	+	+	-	-	-	?	+	-
you	+	-	+	-	-	?	+	-
her	+	-	-	+	-	+	+	-
myself	+	+	-	-	-	?	+	+
herself	+	-	-	+	-	+	+	+
itself	+	-	-	+	-	?	-	+

Figure 3.4. Feature Settings For Some Typical Noun Phrases.

Uplink.

The uplink field of a C-node, S-node, or N-node links to the father node of the C-node, S-node, or N-node in the

C-S-N tree in which it occurs. An example of a C-S-N tree with uplinks shown is given in Figure 3.5.

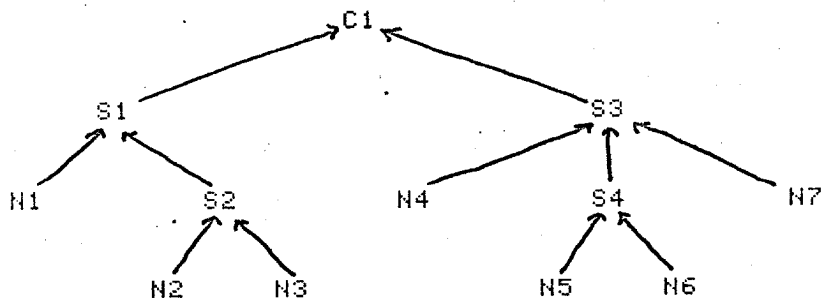


Figure 3.5. An Example Of A C-S-N Tree With Uplinks Shown.

Downlink.

The downlink field of a C-node, S-node, or N-node links to the first son node of the C-node, S-node, or N-node in the C-S-N tree in which it occurs. An example of a C-S-N tree with downlinks shown is given in Figure 3.6.

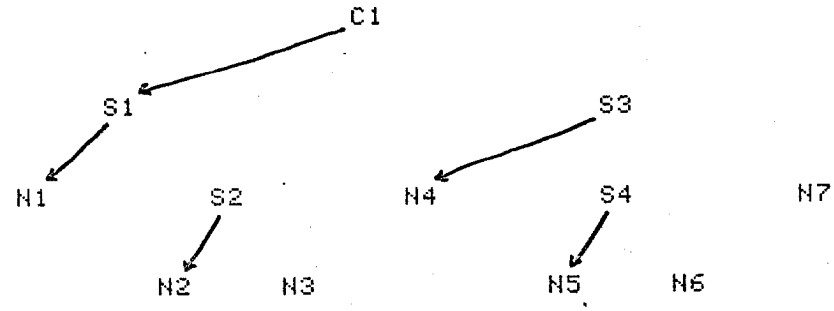


Figure 3.6. An Example Of A C-S-N Tree With Downlinks Shown.

Leftlink.

The leftlink field of a C-node, S-node, or N-node links to the left brother node of the C-node, S-node, or N-node in the C-S-N tree in which it occurs. An example of a C-S-N tree with leftlinks shown is given in Figure 3.7.

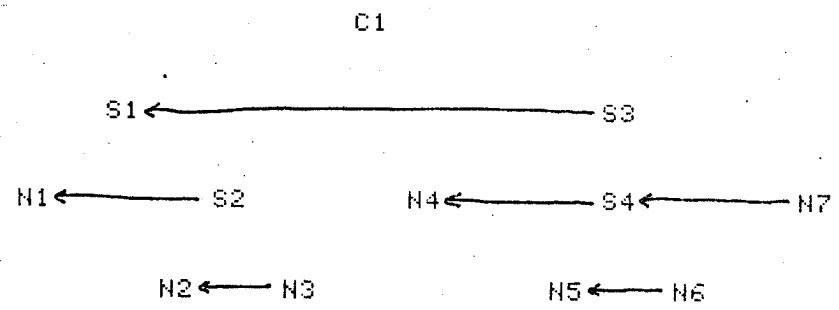


Figure 3.7. An Example Of A C-S-N Tree With Leftlinks Shown.

Rightlink.

The rightlink field of a C-node, S-node, or N-node links to the right brother node of the C-node, S-node, or N-node ir the C-S-N tree in which it occurs. An example of a C-S-N tree with rightlinks shown is given in Figure 3.8.

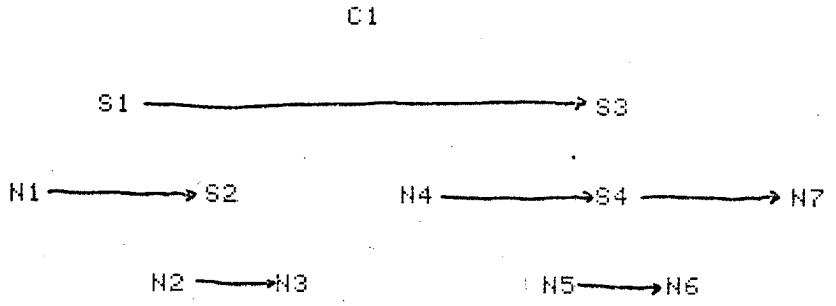


Figure 3.8. An Example Of A C-S-N Tree With Rightlinks

Shown.

Threadlink.

The threadlink field of a C-node, S-node, or N-node links to the first node traversed after the C-node, S-node, or N-node in an inorder traversal of the C-S-N tree in which it occurs. An example of a C-S-N tree with threadlinks shown is given in Figure 3.9.

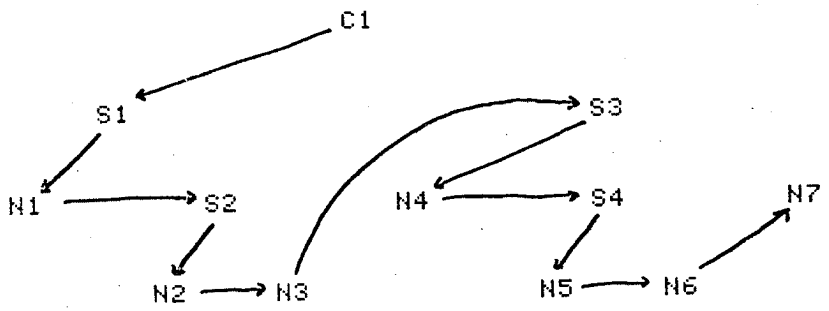


Figure 3.9. An Example Of A C-S-N Tree With Threadlinks Shown.

Number Field.

C-node, S-node, or N-node have a number field which is the number that would be assigned to that node if the nodes of the C-S-N tree in which it occurs are numbered in an inorder traversal. An example of a C-S-N tree with numberfields shown is given in Figure 3.10.

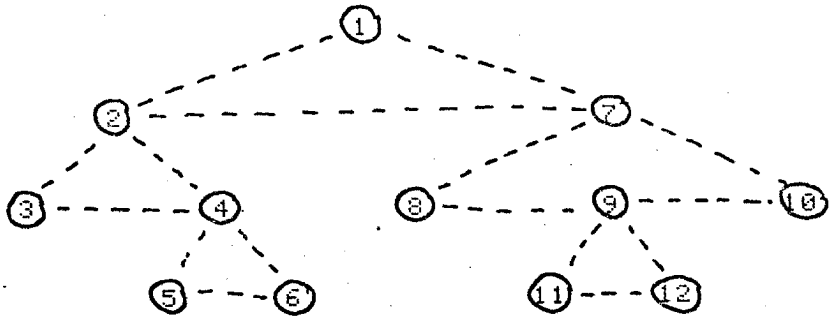


Figure 3.10. An Example Of A C-S-N Tree With Number Fields Shown.

Nplink.

For an E-node, the nplink is the N-node to which it is attached. Conceptually, we think of the E-node as being a copy of the N-node except for its subscript and different set of features, chainlink, and collink. The nplink is just a way of avoiding duplication of information. For an N-node, the nplink is always itself. An example of a chaining table with nplinks shown is given in Figure 3.11.

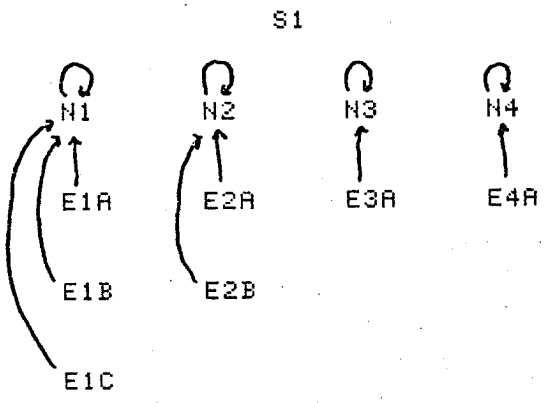


Figure 3.11. An Example Of A Chaining Table With

Nplinks Shown.

Chainlink.

The chainlink of an E-node is another E-node representing the substitute to which the first E-node is attached. When chaining is obligatory, an N-node is chained to an N-node. An example of a chaining table with chainlinks shown is given in Figure 3.12.

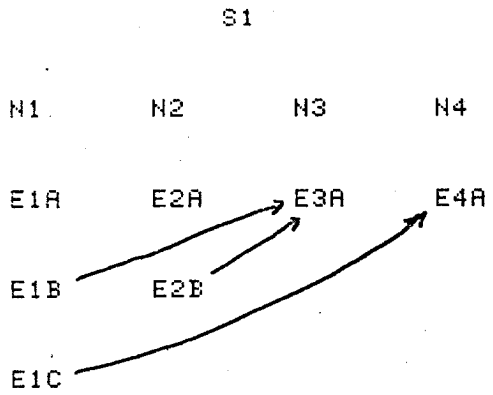


Figure 3.12. An Example Of A Chaining Table With Chainlinks Shown.

Collink.

The collink field of an E-node or N-node links together the elements of a column in a table. An N-node is always on top of a column with E-nodes underneath. An example of a chaining table with collinks shown is given in Figure 3.13.

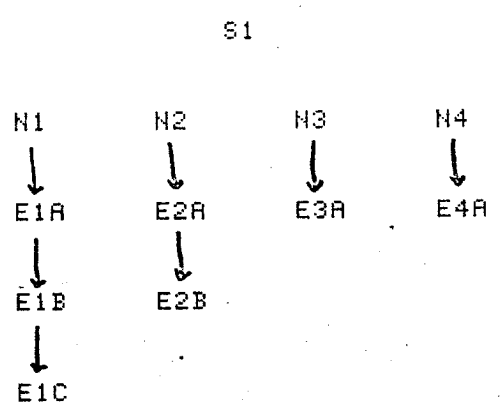


Figure 3.13. An Example Of A Chaining Table With Collinks Shown.

Endcollink.

The endcollink field of an N-node links to the end of the column of E-nodes lying under this N-node. An example of a chaining table with endcollinks shown is given in Figure 3.14.

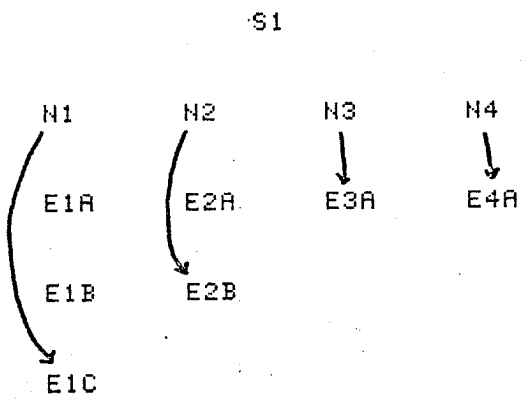


Figure 3.14. An Example Of A Chaining Table With Endcollinks Shown.

Predlink.

The predlink field of an N-node links to the preceding N-node found in an inorder traversal of the C-S-N tree in which it occurs. An example of a C-S-N tree with predlinks shown is given in Figure 3.15.

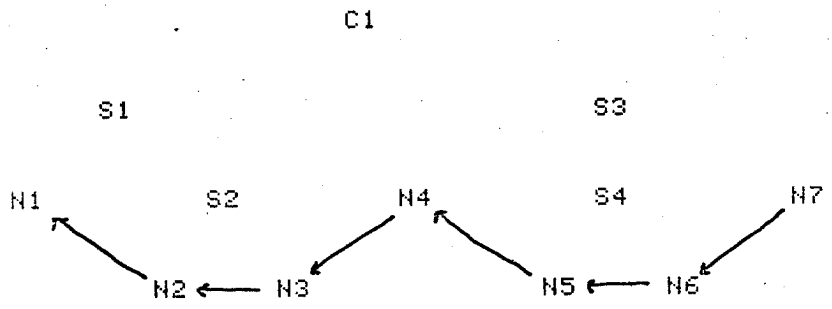


Figure 3.15. An Example Of A C-S-N Tree With Predlinks Shown.

An example of a chaining table with predlinks shown is given in Figure 3.16

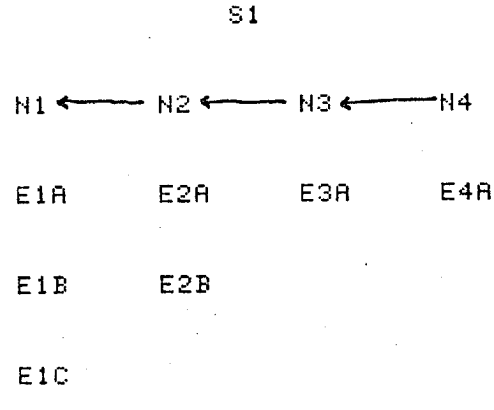


Figure 3.16. An Example Of A Chaining Table With Predlinks Shown.

Succlink.

The succlink field of an N-node links to the succeding N-node found in an inorder traversal of the C-S-N tree in which it occurs. An example of a C-S-N tree with succlinks shown is given in Figure 3.17.

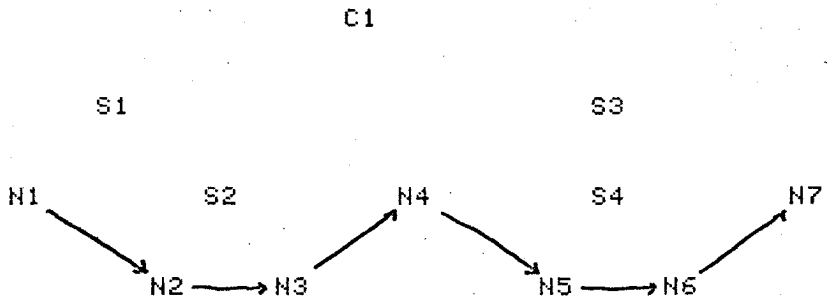


Figure 3.17. An Example Of A C-S-N Tree With Succlinks Shown.

An example of a chaining table with succlinks shown is given in Figure 3.18

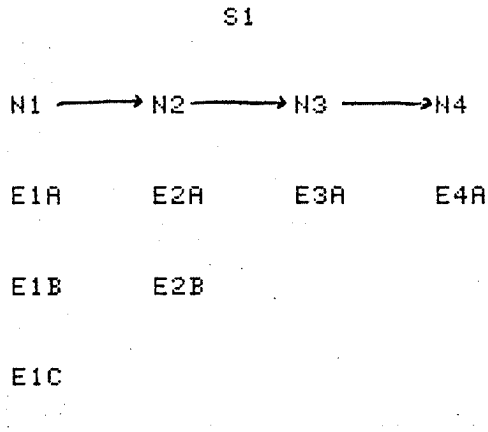


Figure 3.18. An Example Of A Chaining Table With

Succlinks Shown.

Chapter 4. The Node Procesor.

The Node Processor module contains functions Newcnode, Newsnode, Newnnode, and Newenode and has the skeleton shown below in Figure 4.1.

```
module node_processor;
    use globals;
    define function newcnode:node_pointer;
           function newsnode:node_pointer;
           function newnnode:node_pointer;
           function newenode:node_pointer;
    implement
        (*$I NODE_PROC*)
    end;
```

Figure 4.1. Skeleton Of The Node Processor.

Newcnode, Newsnode, Newnnode, and Newenode generate, respectively, a new C-node, S-node, N-node, or E-node, with their fields initialized and are rather straight forward functions. These are shown below in Figures 4.2-4.5.

Function Newcnode returns a new C-node.

```
function newcnode:node_pointer;
    var answer:node_pointer;
begin
    new(answer);
    with answer^ do begin
        id:=cnode;
        uplink:=nil;
        downlink:=nil;
        leftlink:=nil;
        rightlink:=nil;
        threadlink:=nil;
        number:=0;
    end;
    newcnode:=answer;
end;
```

Figure 4.2. Function Newcnode.

Function Newsnode returns a new S-node.

```
function newsnode:node_pointer;
    var answer:node_pointer;
begin
    answer:=newcnode;
    answer^.id:=snode;
    newsnode:=answer;
end;
```

Figure 4.3. Function Newsnode.

Function Newnnode returns a new N-node.

```
function newnnode:node_pointer;
    var answer:node_pointer;
        i:integer;
begin
    new(answer);
    with answer^ do begin
        id:=nnode;
        lit^:='';
        for i:=1 to nfeatures do ftr[i]:=question;
        uplink:=nil;
        downlink:=nil;
        leftlink:=nil;
        rightlink:=nil;
        threadlink:=nil;
        nplink:=nil;
        chainlink:=nil;
        collink:=nil;
        endcollink:=nil;
        predlink:=nil;
        succlink:=nil;
        number:=0;
    end;
    newnnode:=answer;
end;
```

Figure 4.4. Function Newnnode.

Function Newenode returns a new E-node.

```
function newenode:node_pointer;
    var answer:node_pointer;
        i:integer;
begin
    new(answer);
    with answer^ do begin
        id:=enode;
        sub:=' ';
        for i:=1 to nfeatures do ftr[i]:=question;
        nplink:=nil;
        chainlink:=nil;
        collink:=nil;
    end;
    newenode:=answer;
end;
```

Figure 4.5. Function Newenode.

Procedure Listnode.

There is one output procedure in the node processor that has not been discussed above that we need to know about, because we will be looking at some of its output for a short while. This is procedure Listnode which takes as an argument a node_pointer and outputs it in readable form. Otherwise, procedure Listnode does no processing of its own, and so we do not need to know the details of its inner workings. For us it is enough to be able to understand the output. Procedure Listnode has the form indicated in Figure 4.6.

```
procedure listnode(n:node_pointer);
begin
    (output n in a readable form)
end;
```

Figure 4.6. Skeleton Of Procedure Listnode.

Some typical output of procedure Listnode is shown below in Figure 4.7 where a chaining table is listed. (Links from the chaining table to its associated C-S-N tree are also listed by procedure Listnode.)

```
 1:(S, up: 0, dn: 0, lt: 2, rt: 7, th: 0, nu: 0)
 2:(N, lit:       June, ftr:[---+-++-], up: 3, dn: 0,
    lt: 0, rt:10, th:10, np: 2, ch: 0, co:13, ec:14,
    pr: 0, su:10, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 5)
 6:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co: 9, ec: 9,
    pr:10, su: 7, nu: 6)
 7:(N, lit:       them, ftr:[+--++??-], up: 5, dn: 0,
    lt: 6, rt: 0, th: 0, np: 7, ch: 0, co: 8, ec: 8,
    pr: 6, su: 0, nu: 7)
 8:(E, sub:A, ftr:[+--++??-], np: 7, ch: 0, co: 0)
 9:(E, sub:A, ftr:[+--+-++-], np: 6, ch: 0, co: 0)
10:(N, lit:    flowers, ftr:[---++?--], up: 3, dn: 0,
    lt: 2, rt: 0, th: 5, np:10, ch: 0, co:11, ec:12,
    pr: 2, su: 6, nu: 4)
11:(E, sub:A, ftr:[---++?--], np:10, ch: 0, co:12)
12:(E, sub:B, ftr:[+--++???], np:10, ch: 8, co: 0)
13:(E, sub:A, ftr:[---+-++-], np: 2, ch: 0, co:14)
14:(E, sub:B, ftr:[+--+-++?], np: 2, ch: 9, co: 0)
```

Figure 4.7. Typical Output From Procedure Listnode. (C=C-node, S=S-node, N=N-node, E=E-node, lit=lit field, sub=sub field, ftr=ftr field, up=uplink, dn=downlink, lt=leftlink, rt=rightlink, th=threadlink, nu=number, np=nplink, ch=chainlink, co=collink, ec=endcollink, pr=predlink, and su=succlink)

Chapter 5. The Parser.

The Parser module defines function Parse and has the form shown below in Figure 5.1.

```
module parser;
    use globals,node_processor;
    define function parse(sysfocus:node_pointer;
                          systree:system_tree)
                                  :node_pointer;
    implement
        (*$I PARSER*)
    end;
```

Figure 5.1. Skeleton Of The Parser.

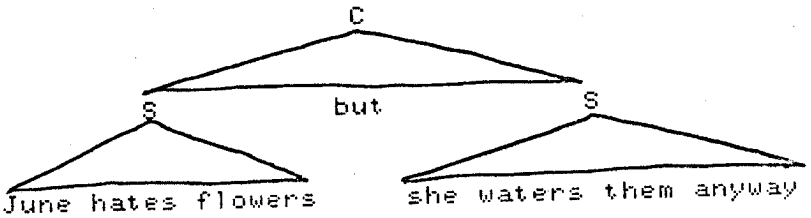
Function Parse accepts as input a system focus representation and system parse tree that has been generated by a computer natural language system. The output of Parse is a C-S-N tree incorporating the information contained in the system parse tree and system focus. The system focus represents the natural language system's focus of attention. This will be gone into in more detail in Chapter 13.

The representation of the system tree inputted to Parse is system dependent, and so the details of Parse are also system dependent. As the internals of Parse are heavily dependent upon and rather involved for any system, we won't go into the details of parse for any particular system here. Hopefully, the reader may glean enough information from the multitude of examples presented in this paper to get an idea

of what Parse does. In any case, lack of an actual algorithm for parse isn't so bad since the ideas presented in this paper are really still in an early stage, and it is enough to concentrate on them.

Even though the input to the parser is not well defined, the output is. The parser builds from the system parse tree it is given the corresponding C-S-N tree with all uplinks, downlinks, leftlinks, rightlinks, threadlinks, and numbers set to what is expected. Consider example (5.2) below.

(5.2) June hates flowers, but she waters them anyway.



When procedure parse is called on the system parse tree representing (5.2), we get the following output in Figure 5.3.

```
PARSE

************************
        FEATURES
************************

            pnf,fpf,spf,tpf,plf,gnf,anf,rpf
      June   -   -   -.  +   -   +   +   -
   flowers   -   -   -   +   +   ?   -   -
       she   +   -   -   +   -   +   +   -
      them   +   -   -   +   +   ?   ?   -
PARSE:exiting

************************
          TREE
************************

 1:(C, up: 0, dn: 2, lt: 0, rt: 0, th: 2, nu: 1)
 2:(S, up: 1, dn: 3, lt: 0, rt: 5, th: 3, nu: 2)
 3:(N, lit:       June, ftr:[---+-++-], up: 2, dn: 0,
    lt: 0, rt: 4, th: 4, np: 3, ch: 0, co: 0, ce: 0,
    pr: 0, su: 0, nu: 3)
 4:(N, lit:    flowers, ftr:[---++?--], up: 2, dn: 0,
    lt: 3, rt: 0, th: 5, np: 4, ch: 0, co: 0, ce: 0,
    pr: 0, su: 0, nu: 4)
 5:(S, up: 1, dn: 6, lt: 2, rt: 0, th: 6, nu: 5)
 6:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co: 0, ce: 0,
    pr: 0, su: 0, nu: 6)
 7:(N, lit:       them, ftr:[+--++??-], up: 5, dn: 0,
    lt: 6, rt: 0, th: 0, np: 7, ch: 0, co: 0, ce: 0,
    pr: 0, su: 0, nu: 7)
```

Figure 5.3. Typical Output From Parse.

Listing of nodes in Figure 5.3 is done by procedure Listnode of the Node Processor described in Chapter 4. The C-S-N parse tree is slightly more complicated when focussing is taken into account, but for the time being we will ignore its effects. We will discuss the effects of focussing on C-S-N parse trees in Chapter 13.

When Figure 5.3 is drawn as a tree, we get a structure

like Figure 5.4.

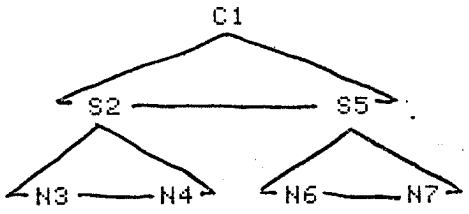


Figure 5.4. Output From Parse Drawn As A Tree.

Chapter 6. Primary Utilitities.

The primary utilities module defines the boolean functions precede, command, and separate corresponding to the precede, command, and separate relations discussed in Chapter 1. The skeleton of the primary utilitites module is shown below in Figure 6.1.

```
module primary_uty;
    use globals;
    define function precede(n1,n2:node_pointer)
                                              :boolean;
           function command(n1,n2:node_pointer)
                                             :boolean;
           function separate(n1,n2:node_pointer)
                                             :boolean;
    implement
        (*$I PRIMARY*)
    end;
```

Figure 6.1. Skeleton Of The Primary Utitlities.

The precede, command, and separate functions do just what is expected. They are true if and only if the precede, command, and separate relations hold between their arguments. Along with function dominate which is used by separate, these functions are shown below in Figures 6.2-6.5.

Function precede is true if and only if n1 precedes n2.

```
function precede(n1,n2:node_pointer):boolean;
begin
    if n1^.id=enode then n1:=n1^.nplink;
    if n2^.id=enode then n2:=n2^.nplink;
    precede:=n1^.number<n2^.number;
end;
```

Figure 6.2. Function Precede.

Function dominate is true if and only if n1 dominates n2.

```
function dominate(n1,n2:node_pointer):boolean;
    label 100;
    var son:node_pointer;
begin
    if n1^.id=enode then n1:=n1^.nplink;
    if n2^.id=enode then n2:=n2^.nplink;
    dominate:=true;
    son:=n1^.downlink;
    while son<>nil do begin
        if son=n2 then goto 100;
        if dominate(son,n2) then goto 100;
        son:=son^.rightlink;
    end;
    dominate:=false;
100:;
end;
```

Figure 6.3. Function Dominate.

Function command is true if and only if n1 commands n2.

```
function command(n1,n2:node_pointer):boolean;
begin
    (minimum cyclic node dominating n1 also dominates n2)
    if n1^.id=enode then n1:=n1^.nplink;
    if n2^.id=enode then n2:=n2^.nplink;
    command:=dominate(n1^.uplink,n2);
end;
```

Figure 6.4. Function Command.

Function separate is true if and only if n1 is separate from n2.

```
function separate(n1,n2:node_pointer):boolean;
    var father:node_pointer;
begin
    {minimum cyclic node dominating n1 that also dominates
     n2 is a C-node}
    if n1^.id=enode then n1:=n1^.nplink;
    if n2^.id=enode then n2:=n2^.nplink;
    father:=n1^.uplink;
    while not dominate(father,n2) do father:=father^.uplink
    separate:=father^.id=cnode;
end;
```

Figure 6.5. Function Separate.

Chapter 7. Secondary Utilities.

The Secondary Utiltitjes module defines functions Sc, Agr, and Rnr. These stand for Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule. The skeleton of the Secondary Utilities module is shown below in Figure 7.1.

```
module secondary_uty;
    use globals,primary_uty;
    define function sc(n1,n2:node_pointer):boolean;
           function agr(n1,n2:node_pointer):boolean;
           function rnr(n1,n2:node_pointer):boolean;
    implement
        (*$I SECONDARY*)
    end;
```

Figure 7.1. Skeleton Of Secondary Utilities.

Syntactic Conditions.

As shown in Chapter 1, certain constraints such as the precede and command rule apply in forward pronominalization. Function Sc is true whenever these grosser syntactic constraints are met. In this paper, we let Sc be true when the precede and command rule is satisfied. Function Sc is shown below in Figure 7.2.

```
function sc(n1,n2:node_pointer):boolean;
(Syrtactic Conditions)
begin
    sc:=not (precede(n1,n2) and (command(n1,n2)
             or separate(n1,n2)));
end;
```

Figure 7.2. Function Sc (Syntactic Conditions).

Agreement.

Besides satisfying Syntactic Conditions, there has to be agreement between a node and its chaining node. First person, second person, third person, plural, gender, and animate features have to agree in order for one node to chain to another. Function Agr is shown below in Figure 7.3.

```
function agr(n1,n2:node_pointer):boolean;
(Agreement)
    var ftr1,ftr2:features;
begin
    ftr1:=n1^.ftr;
    ftr2:=n2^.ftr;
    agr:=eqfeat(ftr1[fpf],ftr2[fpf]) and
         eqfeat(ftr1[spf],ftr2[spf]) and
         eqfeat(ftr1[tpf],ftr2[tpf]) and
         eqfeat(ftr1[plf],ftr2[plf]) and
         eqfeat(ftr1[gnf],ftr2[gnf]) and
         eqfeat(ftr1[anf],ftr2[anf]);
end;
```

Figure 7.3. Function Agr (Agreement).

Function Agr uses function Eqfeat. Eqfeat tests if two features are equal. As indicated in Chapter 1, features are equal unless a plus and minus are compared. Function eqfeat is shown below in Figure 7.4.

```
function eqfeat(x1,x2:feature):boolean;
(Equal Features)
begin
    case x1 of
        plus:     eqfeat:=x2<>minus;
        minus:    eqfeat:=x2<>plus;
        question: eqfeat:=true;
    end;
end;
```

Figure 7.4. Function Eqfeat (Equal Features).

The Reflexive Nonreflexive Rule.

The distinction between reflexive pronouns and nonreflexive pronouns is that reflexive pronouns cannot chain to an N-node that is outside of the same simplex in which it occurs, while a nonreflexive pronoun can. This rule will have to be modified later for genitives, but for now we can suppose that a nonreflexive pronoun must chain to an N-node outside of the same simplex in which it is in. Shown in Figure 7.5 is function Rnr which is true when the reflexive nonreflexive rule is satisfied.

```
function rnr(n1,n2:node_pointer):boolean;
(Reflexive Nonreflexive Rule)
    var ftr1:features;
begin
    n1:=n1^.nplink;
    n2:=n2^.nplink;
    ftr1:=n1^.ftr;
    case ftr1[rpf] of'
        plus:     rnr:=n1^.uplink=n2^.uplink;
        question: (doesn't occur);
        minus:    rnr:=n1^.uplink<>n2^.uplink;
    end;
end;
```

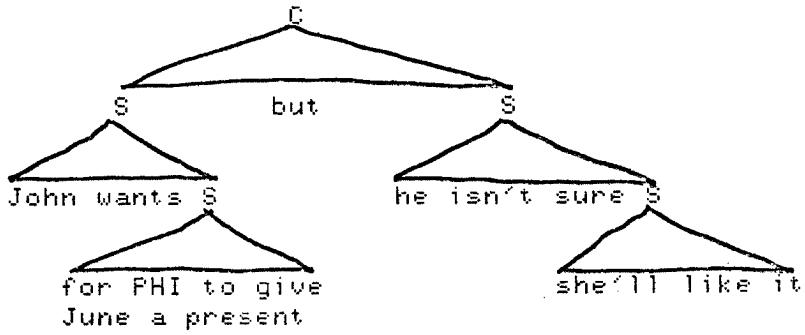
Figure 7.5. Function Rnr (Reflexive Nonreflexive Rule).

Chapter 8. The Table Processor I.

The Table Processor module defines function Chaining which takes as input a C-S-N tree and returns its chaining table. The actions of function Chaining in the Table Processor can only be understood by example, and this is what this chapter provides. In Chapter 9 we'll look at the actual algorithms and in Chapter 10 we'll look at some actual output.

So, let us consider sentence (8.1) below.

(8.1) John wants to give June a present, but he isn't sure she'll like it.



The Parser builds from the system parse tree of (8.1) the corresponding C-S-N tree with six N-nodes which have the lit fielcs and features indicated below in Figure 8.2.

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
John	-	-	-	+	-	-	+	-
PHI	+	?	?	?	?	?	?	-
June	-	-	-	+	-	+	+	-
a present	-	-	-	+	-	?	-	-
he	+	-	-	+	-	-	+	-
she	+	-	-	+	-	+	+	-
it	+	-	-	+	-	?	-	-

Figure 8.2. Lit Fields And Features Of The N-Nodes.

The C-S-N tree itself has the form of Figure 8.3 below.

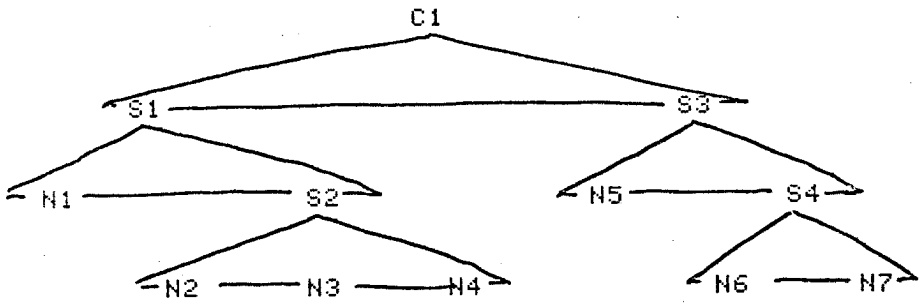


Figure 8.3. C-S-N Parse Tree.

After Parse is called, Chaining is called. The first thing to happen is the initialization of the chaining table for the C-S-N tree. Below each N-node is suspended, by the collink of the N-node, a new E-node with subscript A. Each new E-node has nplink back to the N-node it is suspended from. As well, the features of each new E-node are copied from the N-node it is suspended from. Attached to the first and last N-nodes is an S-node to make it easy to keep track of the first and last N-nodes in the chaining table. The chaining table, as it looks immediately after

initialization, is shown below in Figure 8.4.

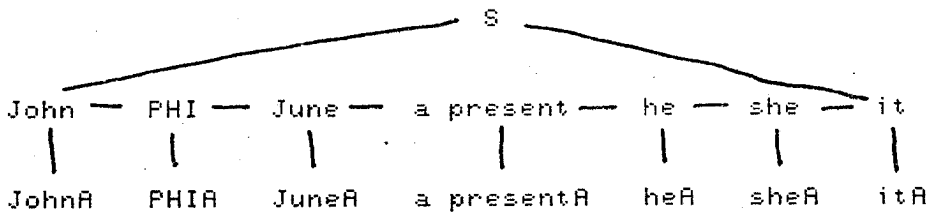


Figure 8.4. Chaining Table Immediately After Initialization.

The chaining algorithm works by walking backwards across N-nodes in the top row and walking down columns of E-nodes. The chaining algorithm works on two N-nodes at a time. If the first is compatible with the second under Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule, then the E-nodes underneath the first N-node that agree with the second N-node are chainlinked to copies of the second N-node.

The last N-node in the table is it, the chaining table begins with it. It can't chain to itself, so the second N-node in the description above becomes she and the chaining algorithm compares it to she. Syntactic Conditions are satisfied, but Agreement isn't.

SC(it, she)=true
AGR(it, she)=false

The chaining algorithm now moves from she to he and compares it to he. Again Syntactic Conditions are satisfied, but Agreement isn't.

SC(it, he)=true
AGR(it, he)=false

The chaining algorithm moves from he to a present and compares it to a present. This time, Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule are satisfied.

SC(it, a present)=true
AGR(it, a present)=true
RNR(it, a present)=true

Since all three rules are satisfied, a chain from itA to a copy of a present may be created. This happens if itA and a present agree, and they do.

AGR(itA, a present)=true

The chaining algorithm makes a new E-node copy of a present, a presentB, and hangs it below a present. The chainlink of a presentB is set to itA and the semantic features of itA, but not the syntactic features, are copied into the semantic features of a presentB. After chaining a presentB to itA, the chaining table appears as shown in Figure 8.5.

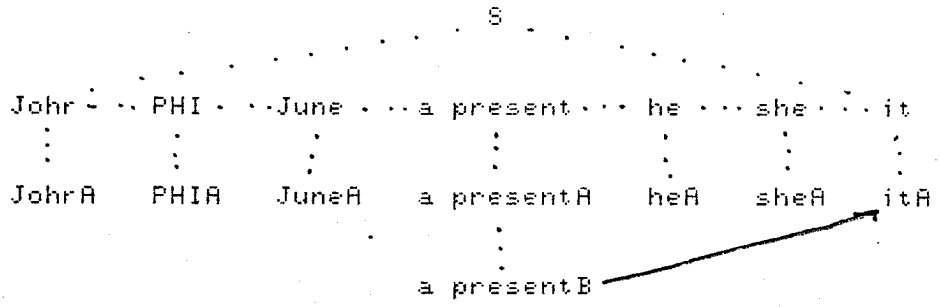


Figure 8.5. Chaining Table After Chaining A PresentB To ItA.

The chaining algorithm now moves from a present to June. Syntactic Conditions are satisfied, but Agreement isn't.

SC(it, June)=true
AGR(it, June)=false

The chaining algorithm now moves from June to PHI. This time, all three rules, Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule are satisfied.

SC(it, PHI)=true
AGR(it, PHI)=true
RNR(it, PHI)=true

Since all three rules are satisfied, E-nodes under it that agree with PHI chain to copies of PHI. ItA is compared to PHI, and it is seen that they agree.

AGR(itA, PHI)=true

The chairing algorithm makes a new E-node copy of PHI, PHIB, and hangs it below PHI. The chainlink of PHIB is set to itA and the semantic features of itA are copied into the semantic features of PHIB. After chaining PHIB to itA, the chaining table appears as shown in Figure 8.6.

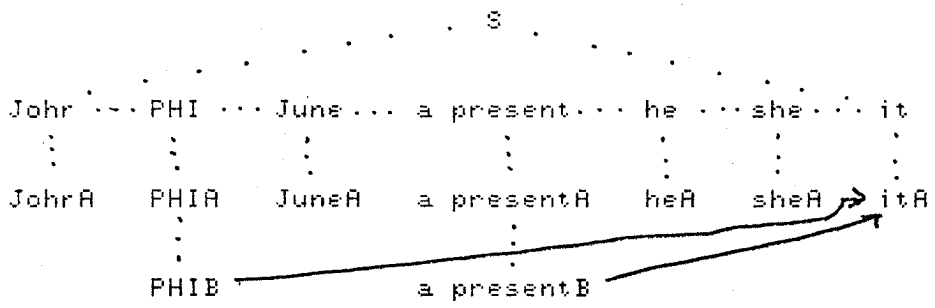


Figure 8.6. Chaining Table After Chaining PHIB To ItA.

The chaining algorithm now moves to John and compares John to it. Syntactic Conditions hold, but Agreement does not.

SC(it, John)=true
AGR(it, John)=false

Having exhausted all possible combinations with it, the chaining algorithm considers she.

The chaining algorithm tries comparing she to it, but Syntactic Conditions are not satisfied.

SC(she, it)=false

The chaining algorithm moves from it to she, but she

can't chain to she, so the chaining algorithm moves to he. This time Syntactic Conditions are satisfied, but Agreement isn't.

SC(she, he)=true
AGR(she, he)=false

The chaining algorithm now moves from he to a present, where again Syntactic Conditions are satisfied, but Agreement isn't.

SC(she, a present)=true
AGR(she, a present)=false

The chaining algorithm moves from a present to June. This time all three rules are satisfied.

SC(she, June)=true
AGR(she, June)=true
RNR(she, June)=true

As all three rules are satisfied, E-nodes under she that agree with June chain to copies of June. SheA is compared to June, and it is seen that they agree.

AGR(sheA, June)=true

The chaining algorithm makes a new E-node copy of June, JuneB, and hangs it below June. The chainlink of JuneB is set to sheA and the semantic features of sheA are copied into the semantic features of JuneB. After chaining JuneB to sheA, the chaining table appears as shown in Figure 8.7.

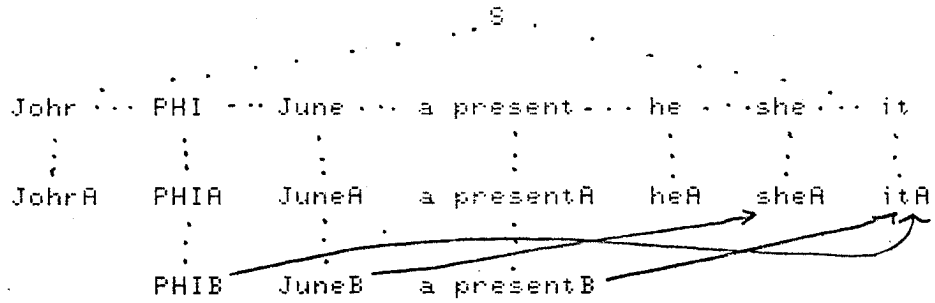


Figure 8.7. Chaining Table After Chaining JuneB To SheA.

The chaining algorithm now moves from June to PHI and compares she to PHI. All three rules are satisfied.

SC(she, PHI)=true
AGR(she, PHI)=true
RNR(she, PHI)=true

Copies of PHI are chainlinked to E-nodes under she that agree with PHI. SheA is compared to PHI, and it is seen that they agree.

AGR(sheA, PHI)=true

A new E-rode copy of PHI, PHIC, is made and hung below PHI. The chairlink of PHIC is set to sheA and the semantic features of sheA are copied into the semantic features of PHIC. Ffter chaining PHIC to sheA, the chaining table appears as shown in Figure 8.8.

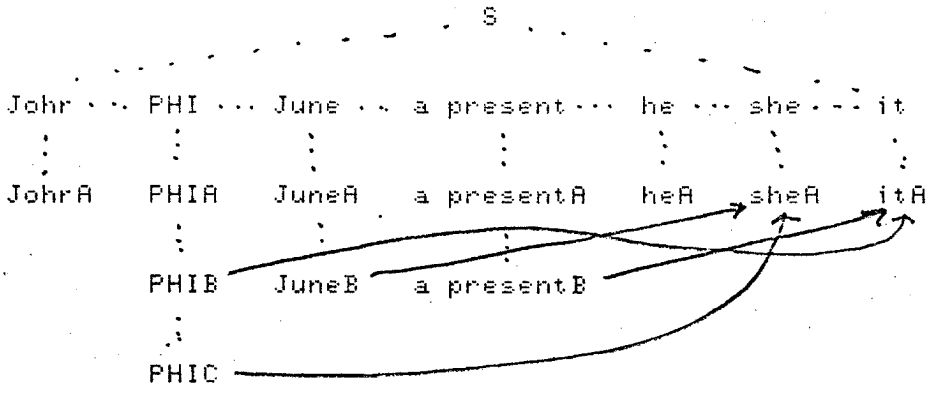


Figure 8.8. Chaining Table After Chaining PHIC To SheA.

The chaining algorithm now moves from PHI to John and compares she to John. It is seen that Syntactic Conditions are satisfied, but Agreement isn't.

SC(she, John)=true
AGR(she, John)=false

This completes the creation of chainlinks to E-nodes under she. The chaining algorithm now considers he.

He is compared to it, but it is seen that Syntactic Conditions aren't satisfied.

SC(he, it)=false

The chaining algorithm moves from it to she, but again, Syntactic Conditions aren't satisfied.

SC(he, she)=false

The chaining algorithm moves from she to he, but he can't chain to he, so the chaining algorithm moves from he to a present. This time Syntactic Conditions are satisfied,

but Agreement isn't.

SC(he, a present)=true
AGR(he, a present)=false

The chaining algorithm moves from a present to June, and similar results happen.

SC(he, June)=true
AGR(he, June)=false

Next, the chaining algorithm moves from June to PHI, and this time all three rules, Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule are satisfied.

SC(he, PHI)=true
AGR(he, PHI)=true
RNR(he, PHI)=true

Copies of he are chainlinked to E-nodes under PHI that agree with he. HeA is compared to PHI, and it is seen that they agree.

AGR(heA, PHI)=true

A new E-rode copy of PHI, PHID, is made and hung below PHI. The chairlink of PHID is set to heA and the semantic features of heA are copied into the semantic features of PHID. Ffter chaining PHID to heA, the chaining table appears as shown in Figure 8.9.

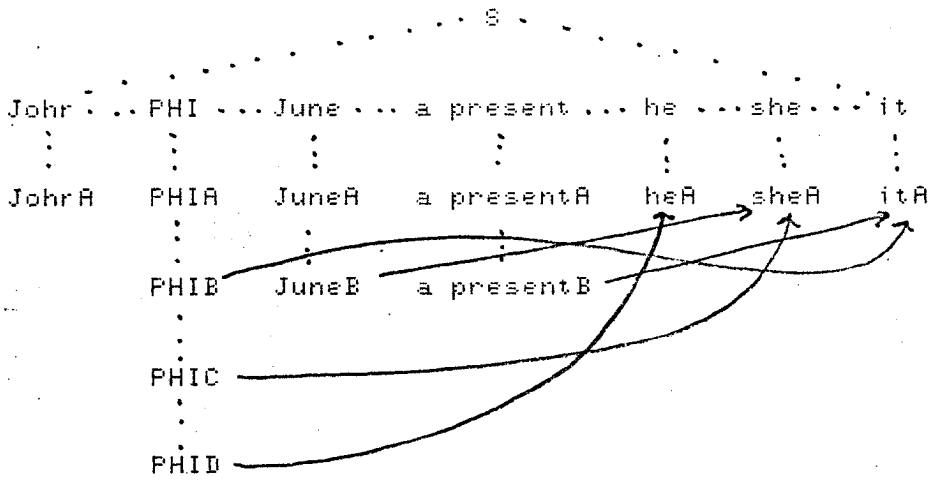


Figure 8.9. Chaining Table After Chaining PHID To HeA.

The chaining algorithm now moves from PHI to John and he is cmcpared John. It is seen that all three rules are satisfiec.

SC(he, John)=true
AGR(he, John)=true
RNR(he, John)=true

So, copies of he are chainlinked to E-nodes under John that agree with he. HeA is compared to John, and it is seen that they agree.

AGR(heA, John)=true

A new E-rode copy of John, JohnB, is made and hung below John. The chainlink of JohnB is set to heA and the semantic features of heA are copied into the semantic features of JohnB. After chaining JohnB to heA, the chaining table appears as shown in Figure 8.10.

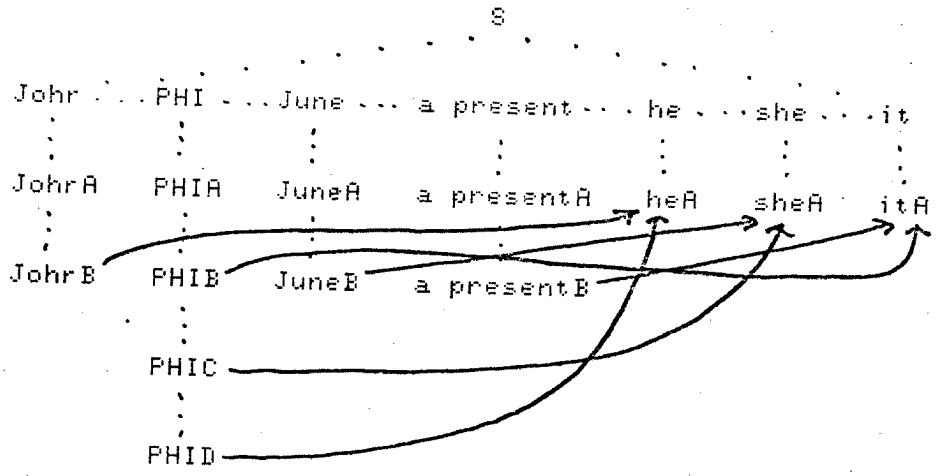


Figure 8.10. Chaining Table After Chaining JohnB To HeA.

Having completed the processing of he, the chaining algorithm considers a present. A present is not a pronoun though, so the chaining algorithm moves on to June. Similarly, June is not a pronoun, so the chaining algorithm now considers PHI.

The chaining algorithm compares PHI to it, and it is seen that Syntactic Conditions don't hold.

SC(PHI, it)=false

The chaining algorithm moves from it to she, she to he, he to a present, and a present to June with little more success.

SC(PHI, she)=false
SC(PHI, he)=false
SC(PHI, a present)=false
SC(PHI, June)=false

The chaining algorithm moves from June to PHI, but PHI

can't chain to PHI. So now, the chaining algorithm moves from PHI to John. This time, all three rules are satisfied.

SC(PHI, John)=true
AGR(PHI, John)=true
RNR(PHI, John)=true

Copies of John are chainlinked to E-nodes under PHI that agree with John. PHIA is compared to John, and it is seen that they agree.

AGR(PHIA, John)=true

PHIB is compared to John, and it is seen that they don't agree.

AGR(PHIB, John)=false

PHIB and John don't agree because when PHIB was chainlinked to itA, the semantic features of itA were copied into the semantic features of PHIB. Hence, the information that itA was inanimate was copied into PHIB preventing a ridiculous chain: JohnX is chained to PHIB is chained to itA. PHIC, which was chained to sheA, is compared to John, and it is seen that they don't agree.

AGR(PHIC, John)=false

On the other hand, PHID, which was chained to heA, does agree with John.

AGR(PHID, John)=true

Thus, two new copies of John, JohnC and JohnD are made. JohnC is chainlinked to PHIA and JohnD is chainlinked to PHID. The semantic features of PHIA are copied into JohnC,

and the semantic features of PHID are copied into the semantic features of JohnD. After chaining JohnC to PHIA and JohnI to PHID, the chaining table appears as shown in Figure 8.11.

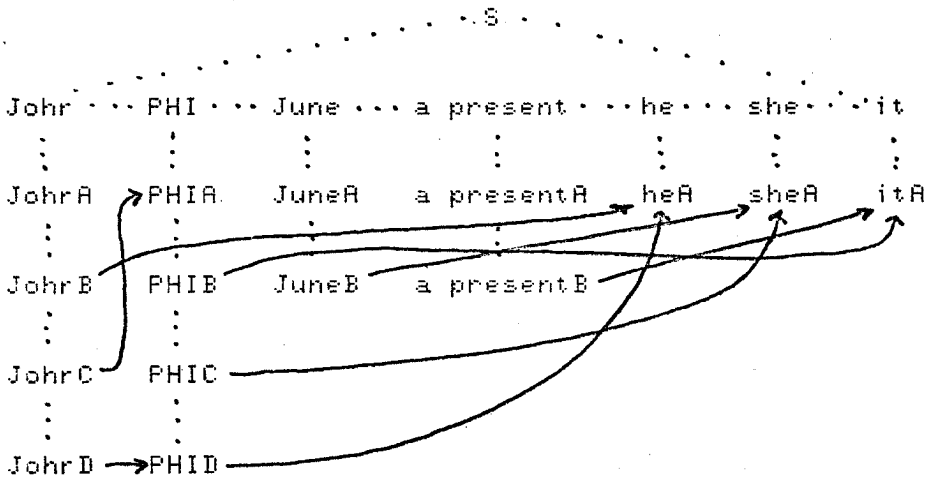


Figure 8.11. Chaining Table After Chaining JohnC To PHIA and JohnD to PHID.

Having completed chaining to PHI, the chaining algorithm moves to John. John is not a pronoun, so the chaining algorithm now stops as it has reached the end of the chairing table. This makes Figure 8.11, above, the finished chaining table.

Chapter 9. Table Processor II.

From Chapter 8 we know that the Table Processor module defines function Chaining which takes as input a C-S-N tree and which returns as output the chaining table of the inputted C-S-N tree. In Chapter 8, we illustrated the kind of processing the Table Processor does by working through in detail a typical example. In this chapter, we will go into the particulars of the Table Processor algorithms. In Chapter 10, we'll look at some actual output.

The skeleton of the Table Processor module is shown below in Figure 9.1. The Table Processor module defines function Chaining.

```
module table_processor;
    use globals,secondary_uty,node_processor;
    define function chaining(tree:node_pointer)
                                        :node_pointer;
    implement
        (*$I TABLE_PROC*)
    end;
```

Figure 9.1. Skeleton Of The Table Processor.

Function Chaining is the algorithm we described by example in Chapter 8. Chaining takes as input a C-S-N tree and returns the chaining table of the inputted C-S-N tree. Function Chaining is shown below in Figure 9.2.

```
function chaining(tree:node_pointer):node_pointer;
    var n1:node_pointer;
begin
    inittable(tree);
    n1:=table^.rightlink;
    while n1<>nil do with n1^ do begin
        if ftr[pnf]=plus then chainingn(n1);
        n1:=predlink;
    end;
    chaining:=table;
end;
```

Figure 9.2. Function Chaining.

The first thing function Chaining does is to call Inittable which initializes the chaining table as described in the previous chapter. Procedure Inittable is shown in Figure 9.3.

```
procedure inittable(tree:node_pointer);
    var first,last,n:node_pointer;
        i:integer;
begin
    first:=nil;
    last:=nil;
    n:=tree;
    while n<>nil do with n^ do begin
        if id=nnode then begin
            collink:=newenode;
            with collink^ do begin
                for i:=1 to nfeatures do
                ftr[i]:=n^.ftr[i];
                nplink:=n;
                sub:='A';
            end;
            endcollink:=collink;
            predlink:=last;
            if last=nil then first:=n
            else last^.succlink:=n;
            last:=n;
        end;
        n:=threadlink;
    end;
    table:=newsnode;
    with table^ do begin
        leftlink:=first;
        rightlink:=last;
    end;
end;
```

Figure 9.3. Procedure Inittable.

Below each N-node is hung a new E-node with features copied from the N-node and nplink back to the N-node. The collinks and endcollinks of the N-nodes are updated accordingly. Predlinks and succlinks are set in Inittable using the threadlinks which were established by the Parser. Finally, at the end of the procedure, table, a variable global inside the Table Processor, has its leftlink and rightlink set to the first and last N-node.

For each N-node that is a pronoun, function Chaining calls procedure Chainingn. Chainingn calls Reflchaining or Nonreflchaining depending on whether or not the inputted N-node is reflexive or not. Procedure Chainingn is shown below in Figure 9.4.

```
procedure chainingn(n1:node_pointer);
begin
    case n1^.ftr[rpf] of
        plus:     reflchaining(n1);
        question: {can't happen};
        minus:    nonreflchaining(n1);
    end;
end;
```

Figure 9.4. Procedure Chainingn.

Procedure Nonreflchaining handles nonreflexive pronouns. Procedure Nonreflchaining is shown below in Figure 9.5.

```
procedure nonreflchaining(n1:node_pointer);
{Nonreflexive Chaining}
    var n2:node_pointer;
begin
    {try nps other than n1}
    n2:=table^.rightlink;          {last N-node}
    while n2<>nil do begin
        if n2<>n1 then chainingnton(n1,n2);
        n2:=n2^.predlink;
    end;
end;
```

Figure 9.5. Procedure Nonreflchaining.

Nonreflchaining calls Chainingnton on the inputted N-node with each N-node in the chaining table except itself.

This takes care of creating all chains to E-nodes lying under the inputted N-node.

Procedure Reflchaining is very similar to Nonreflchaining and is shown below in Figure 9.6.

```
procedure reflchaining(n1:node_pointer);
    var n2:node_pointer;
begin
    (try nps preceding n1 in same simplex)
    n2:=spred(n1);
   ,while n2<>nil do begin
        if n2<>n1 then chainingnton(n1,n2);
        n2:=spred(n2);
    end;
end;
```

Figure 9.6. Procedure Reflchaining.

Since the N-node inputted to Reflchaining is reflexive, Reflchaining only calls Chainingnton on the inputted N-node with each preceding N-node within the same simplex as the inputted N-node.

Function Spred, which is used by procedure Reflchaining, simply returns the N-node that preceeds the inputted N-node in the same simplex. Function Spred is shown below in Figure 9.7.

```
function spred(n1:node_pointer):node_pointer;
    label 100;
    var answer:node_pointer;
begin
    answer:=n1;
    while true do with answer^ do begin
        answer:=leftlink;
        if answer=nil then goto 100;
        if id=nnode then goto 100;
    end;
100:spred:=answer;
end;
```

Figure 9.7. Function Spred.

Procedure Chainingnton is called by procedures Reflchaining and Nonreflchaining and is shown below in Figure 9.7.

```
procedure chainingnton(n1,n2:node_pointer);
    label 100;
    var oldendcollink,e1:node_pointer;
begin
    if not sc(n1,n2) then goto 100;
    if not agr(n1,n2) then goto 100;
    if not rnr(n1,n2) then goto 100;
    oldendcollink:=n1^.endcollink;
    e1:=n1;
    while e1<>oldendcollink do begin
        e1:=e1^.collink;
        if e1<>nil then chainingeton(e1,n2);
    end;
100:;
end;
```

Figure 9.7. Procedure Chainingnton.

If Syntactic Conditions, Agreement, and the Reflexive Nonreflexive Rule hold, then procedure Chainingeton is called on each E-node lying underneath the first N-node.

Procedure Chainingeton, which is called by procedure

Chainingnton, is shown below in Figure 9.8.

```
procedure chainingeton(e1,n2:node_pointer);
begin
    if agr(e1,n2) then newchain(e1,n2);
end;
```

Figure 9.8. Procedure Chainingeton.

If the inputted E-node agrees with the inputted N-node, the a new chain is created from a copy of the inputted N-node to the inputted E-node by calling procedure Newchain.

Procedure Newchain, which is called by Chainingeton, is shown below in Figure 9.9.

```
procedure newchain(e1,n2:node_pointer);
    var n:node_pointer;
        i:integer;
begin
    (add n2^e1, with right subscript and features)
    n:=newenode;
    with n^ do begin
        nplink:=n2;
        chainlink:=e1;
        sub:=chr(ord(n2^.endcollink^.sub)+1);

        (n2 ? nonsyntactic features replaced by
         corresponding e1 features)
        for i:=1 to nfeatures do begin
            if (ftr[i]=question) and (i<>rpf)
            then ftr[i]:=e1^.ftr[i];
        end;
    end;
    with n2^ do begin
         endcollink^.collink:=n;
         endcollink:=n;
    end;
end(newchain);
```

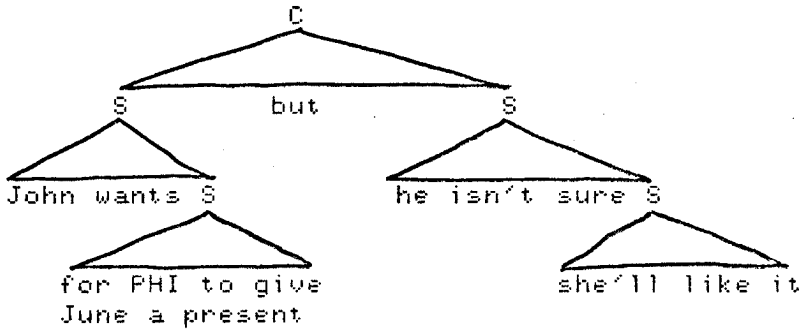
Figure 9.9. Procedure Newchain.

Procedure Newchain creates a copy of the inputted N-node and chainlinks it to the inputted E-node. Semantic features are copied from the inputted E-node to the copy of the inputted N-node.

Chapter 10. Table Processor III.

The last two chapters have been devoted to describing the table processor. It is time for some real examples. The first example we present in this chapter was produced using procedure listnode of the node processor along with some intermittent write statements indicating when we are entering and exiting some of the more important routines and some or their results. We start off with the example first presentec in Chapter 8.

(10.1) John wants to give June a present, but he isn't sure she'll like it.



Processing (10.1) with some intermediate output gives the listing shown below. This is somewhat verbose, but later examples will have will be cleaner and shorter, though less detailed output.

```
PARSE

************************
        FEATURES
************************

            pnf,fpf,spf,tpf,plf,gnf,anf,rpf
      John   -   -   -   +   -   -   +   -
       PHI   +   ?   ?   ?   ?   ?   ?   -
      June   -   -   -   +   -   +   +   -
 a present   -   -   -   +   -   ?   -   -
        he   +   -   -   +   -   -   +   -
       shé   +   -   -   +   -   +   +   -
        it   +   -   -   +   -   ?   -   -
PARSE:exiting

************************
          TREE
************************

 1:(C, up: 0, dn: 2, lt: 0, rt: 0, th: 2, nu: 1)
 2:(S, up: 1, dn: 3, lt: 0, rt: 8, th: 3, nu: 2)
 3:(N, lit:       John, ftr:[---+--+-], up: 2, dn: 0,
    lt: 0, rt: 4, th: 4, np: 3, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu: 3)
 4:(S, up: 2, dn: 5, lt: 3, rt: 0, th: 5, nu: 4)
 5:(N, lit:        PHI, ftr:[+?????-], up: 4, dn: 0,
    lt: 0, rt: 6, th: 6, np: 5, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu: 5)
 6:(N, lit:       June, ftr:[---+-++-], up: 4, dn: 0,
    lt: 5, rt: 7, th: 7, np: 6, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu: 6)
 7:(N, lit: a present, ftr:[---+-?--], up: 4, dn: 0,
    lt: 6, rt: 0, th: 8, np: 7, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu: 7)
 8:(S, up: 1, dn: 9, lt: 2, rt: 0, th: 9, nu: 8)
 9:(N, lit:         he, ftr:[+--+--+-], up: 8, dn: 0,
    lt: 0, rt:10, th:10, np: 9, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu: 9)
10:(N, lit:        she, ftr:[+--+-++-], up: 8, dn: 0,
    lt: 9, rt:11, th:11, np:10, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu:10)
11:(N, lit:         it, ftr:[+--+-?--], up: 8, dn: 0,
    lt:10, rt: 0, th: 0, np:11, ch: 0, co: 0, ec: 0,
    pr: 0, su: 0, nu:11)
```

```
CHAINING
     INITTABLE

************************
          TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John; ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:19, ec:19,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:18, ec:18,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+?????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:17, ec:17,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:16,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co: 0)
17:(E, sub:A, ftr:[+?????-], np:14, ch: 0, co: 0)
18:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co: 0)
19:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)
```

```
CHAININGN
NONREFLCHAINING
CHAININGNTON
     SC(it, she)=true
     AGR(it, she)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(it, he)=true
     AGR(it, he)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(it, a present)=true
     AGR(it, a present)=true
     RNR(it, a present)=true
          AGR(itA, a present)=true
          NEWCHAIN:a presentB chained to itA
```

```
************************
        TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:20, ec:20,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0; rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:18, ec:19,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:17, ec:17,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:16,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co: 0)
17:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co: 0)
18:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
20:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)

CHAININGNTON:exiting
CHAININGNTON
     SC(it, June)=true
     AGR(it, June)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(it, PHI)=true
     AGR(it, PHI)=true
     RNR(it, PHI)=true
          AGR(itA, PHI)=true
          NEWCHAIN:PHIB chained to itA
```

```
*************************
          TABLE
*************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:21, ec:21,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0; rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:19, ec:20,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:17, ec:18,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:16,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co: 0)
17:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:18)
18:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co: 0)
19:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:20)
20:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
21:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)
```

```
************************
        TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:      John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:22, ec:22,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:        he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:       she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:        it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:20, ec:21,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:      PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:19,
    pr: 2, su:15, nu: 5)
15:(N, lit:     June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co: 0)
20:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:21)
21:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
22:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)
```

```
CHAININGNTON:exiting
CHAININGNTON
     SC(she, PHI)=true
     AGR(she, PHI)=true
     RNR(she, PHI)=true
          AGR(sheA, PHI)=true
          NEWCHAIN:PHIC chained to sheA

************************
          TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:23, ec:23,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:21, ec:22,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:20,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co: 0)
21:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:22)
22:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
23:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)
```

```
CHAININGNTON:exiting
CHAININGNTON
     SC(she, John)=true
     AGR(she, John)=false
CHAININGNTON:exiting
NONREFLCHAINING:exiting
CHAININGN:exiting
CHAININGN
NONREFLCHAINING
CHAININGNTON
     SC(he, it)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(he, she)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(he, a present)=true
     AGR(he, a present)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(he, June)=true
     AGR(he, June)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(he, PHI)=true
     AGR(he, PHI)=true
     RNR(he, PHI)=true
          AGR(heA, PHI)=true
          NEWCHAIN:PHID chained to heA
```

```
*************************
        TABLE
*************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:      John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:24,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:       he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0; rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:      she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:       it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:      PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:     June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co: 0)
```

```
CHAININGNTON:exiting
CHAININGNTON
     SC(he, John)=true
     AGR(he, John)=true
     RNR(he, John)=true
          AGR(heA, John)=true
          NEWCHAIN:JohnB chained to heA

************************
          TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:25,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co:25)
25:(E, sub:B, ftr:[+--+--+?], np: 2, ch:11, co: 0)
```

```
*************************
        TABLE
*************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:25,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co:25)
25:(E, sub:B, ftr:[+--+--+?], np: 2, ch:11, co: 0)
```

```
CHAININGNTON:exiting
NONREFLCHAINING:exiting
CHAININGN:exiting
CHAININGN
NONREFLCHAINING
CHAININGNTON
     SC(PHI, it)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(PHI, she)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(PHI, he)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(PHI, a present)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(PHI, June)=false
CHAININGNTON:exiting
CHAININGNTON
     SC(PHI, John)=true
     AGR(PHI, John)=true
     RNR(PHI, John)=true
          AGR(PHIA, John)=true
          NEWCHAIN:JohnC chained to PHIA
```

```
*************************
          TABLE
*************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:26,
    pr: 0, su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co:25)
25:(E, sub:B, ftr:[+--+--+?], np: 2, ch:11, co:26)
26:(E, sub:C, ftr:[+???????], np: 2, ch:18, co: 0)
```

```
          AGR(PHIB, John)=false
          AGR(PHIC, John)=false
          AGR(PHID, John)=true
          NEWCHAIN:JohnD chained to PHID

************************
          TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:27,
    pr: 0; su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co:25)
25:(E, sub:B, ftr:[+--+--+?], np: 2, ch:11, co:26)
26:(E, sub:C, ftr:[+???????], np: 2, ch:18, co:27)
27:(E, sub:D, ftr:[+--+--+?], np: 2, ch:21, co: 0)
```

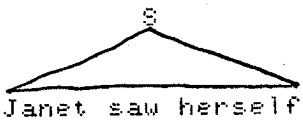
```
CHAININGNTON:exiting
NONREFLCHAINING:exiting
CHAININGN:exiting
CHAINING:exiting

************************
      FINAL TABLE
************************

 1:(S, up: 0, dn: 0, lt: 2, rt: 8, th: 0, nu: 0)
 2:(N, lit:       John, ftr:[---+--+-], up: 3, dn: 0,
    lt: 0, rt:13, th:13, np: 2, ch: 0, co:24, ec:27,
    pr: 0; su:14, nu: 3)
 3:(S, up: 4, dn: 2, lt: 0, rt: 5, th: 2, nu: 2)
 4:(C, up: 0, dn: 3, lt: 0, rt: 0, th: 3, nu: 1)
 5:(S, up: 4, dn: 6, lt: 3, rt: 0, th: 6, nu: 8)
 6:(N, lit:         he, ftr:[+--+--+-], up: 5, dn: 0,
    lt: 0, rt: 7, th: 7, np: 6, ch: 0, co:11, ec:11,
    pr:12, su: 7, nu: 9)
 7:(N, lit:        she, ftr:[+--+-++-], up: 5, dn: 0,
    lt: 6, rt: 8, th: 8, np: 7, ch: 0, co:10, ec:10,
    pr: 6, su: 8, nu:10)
 8:(N, lit:         it, ftr:[+--+-?--], up: 5, dn: 0,
    lt: 7, rt: 0, th: 0, np: 8, ch: 0, co: 9, ec: 9,
    pr: 7, su: 0, nu:11)
 9:(E, sub:A, ftr:[+--+-?--], np: 8, ch: 0, co: 0)
10:(E, sub:A, ftr:[+--+-++-], np: 7, ch: 0, co: 0)
11:(E, sub:A, ftr:[+--+--+-], np: 6, ch: 0, co: 0)
12:(N, lit: a present, ftr:[---+-?--], up:13, dn: 0,
    lt:15, rt: 0, th: 5, np:12, ch: 0, co:22, ec:23,
    pr:15, su: 6, nu: 7)
13:(S, up: 3, dn:14, lt: 2, rt: 0, th:14, nu: 4)
14:(N, lit:       PHI, ftr:[+??????-], up:13, dn: 0,
    lt: 0, rt:15, th:15, np:14, ch: 0, co:18, ec:21,
    pr: 2, su:15, nu: 5)
15:(N, lit:      June, ftr:[---+-++-], up:13, dn: 0,
    lt:14, rt:12, th:12, np:15, ch: 0, co:16, ec:17,
    pr:14, su:12, nu: 6)
16:(E, sub:A, ftr:[---+-++-], np:15, ch: 0, co:17)
17:(E, sub:B, ftr:[+--+-++?], np:15, ch:10, co: 0)
18:(E, sub:A, ftr:[+??????-], np:14, ch: 0, co:19)
19:(E, sub:B, ftr:[+--+-?-?], np:14, ch: 9, co:20)
20:(E, sub:C, ftr:[+--+-++?], np:14, ch:10, co:21)
21:(E, sub:D, ftr:[+--+--+?], np:14, ch:11, co: 0)
22:(E, sub:A, ftr:[---+-?--], np:12, ch: 0, co:23)
23:(E, sub:B, ftr:[+--+-?-?], np:12, ch: 9, co: 0)
24:(E, sub:A, ftr:[---+--+-], np: 2, ch: 0, co:25)
25:(E, sub:B, ftr:[+--+--+?], np: 2, ch:11, co:26)
26:(E, sub:C, ftr:[+???????], np: 2, ch:18, co:27)
27:(E, sub:D, ftr:[+--+--+?], np: 2, ch:21, co: 0)
```

The previous example should give enough details away to satisfy the reader's curiousity, but the form of the previous example is rather burdensome. From now on, we'll keep to a more concise, if less detatiled, output. To indicate a chainlink between two E-nodes, we use the symbol ^. Below are some more examples.

(10.2) Janet saw herself.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Janet	-	-	-	+	-	+	+	-
herself	+	-	-	+	-	+	+	+

INITTABLE

TABLE

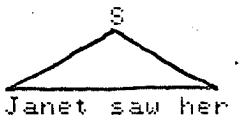
-Janet-	-herself-
JanetA	herselfA

SC(herself, Janet)=true
AGR(herself, Janet)=true
RNR(herself, Janet)=true
AGR(herselfA, Janet)=true
NEWCHAIN:JanetB^herselfA

TABLE

-Janet-	-herself-
JanetA	herselfA
JanetB^herselfA	

(10.3) Janet saw her.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Janet	-	-	-	+	-	+	+	-
her	+	-	-	+	-	+	+	-

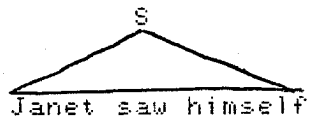
INITTABLE

TABLE

-Janet-	-her-
JanetA	herA

SC(her, Janet)=true
AGR(her, Janet)=true
RNR(her, Janet)=false

(10.7) *Janet saw himself.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Janet	-	-	-	+	-	+	+	-
himself	+	-	-	+	-	-	+	+

INITTABLE

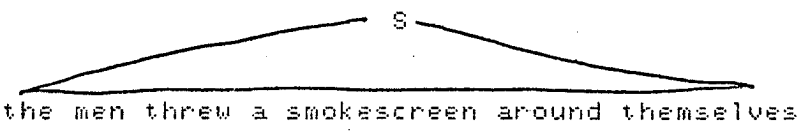
TABLE

-Janet-	-himself-
JanetA	himselfA

SC(himself, Janet)=true
AGR(himself, Janet)=false

Examples (10.5)-(10.11) are from Lees and Klima [22].

(10.5) The men threw a smokescreen around themselves.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
the men	-	-	-	+	+	-	+	-
a smokescr	-	-	-	+	-	?	-	-
themselves	+	-	-	+	+	?	?	+

INITTABLE

TABLE

-the men- the menA	-a smokescreen- a smokescreenA
.-themselves- themselvesA	

SC(themselves, a smokescreen)=true
AGR(themselves, a smokescreen)=false

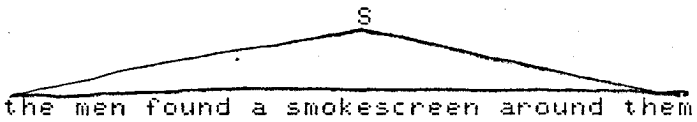
SC(themselves, the men)=true
AGR(themselves, the men)=true
RNR(themselves, the men)=true
AGR(themselvesA, the men)=true
NEWCHAIN:the menB^themselvesA

TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themselvesA	

-themselves-
themselvesA

(10.6) The men found a smokescreen around them.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
the men	-	-	-	+	+	-	+	-
a smokescr	-	-	-	+	-	?	-	-
them	+	-	-	+	+	?	?	-

INITTABLE

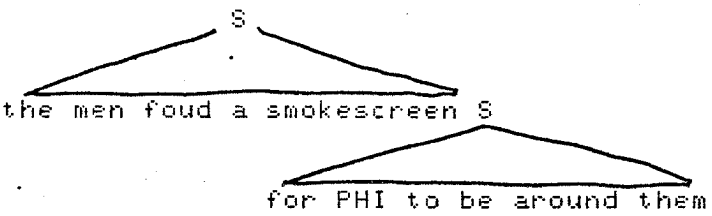
TABLE

-the men- the menA	-a smokescreen- a smokescreenA
-them- themA	

SC(them, a smokescreen)=true
AGR(them, a smokescreen)=false

SC(them, the men)=true
AGR(them, the men)=true
RNR(them, the men)=false

(10.7) The men found a smokescreen to be around them.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
the men	-	-	-	+	+	-	+	-
a smokescr	-	-	-	+	-	?	-	-
PHI	+	?	?	?	?	?	?	-
them	+	-	-	+	+	?	?	-

INITTABLE

TABLE

-the men- the menA	-a smokescreen- a smokescreenA
-PHI- PHIA	-them- themA

SC(them, PHI)=true
AGR(them, PHI)=true
RNR(them, PHI)=false

SC(them, a smokescreen)=true
AGR(them, a smokescreen)=false

SC(them, the men)=true
AGR(them, the men)=true
RNR(them, the men)=true
AGR(themA, the men)=true
NEWCHAIN:the menB^themA

TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themA	

-PHI-	-them-
PHIA	themA

SC(PHI, them)=false

SC(PHI, a smokescreen)=true
AGR(PHI, a smokescreen)=true
RNR(PHI, a smokescreen)=true
AGR(PHIA, a smokescreen)=true
NEWCHAIN:a smokescreenB^PHIA

TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themA	a smokescreenB^PHIA

-PHI-	-them-
PHIA	themA

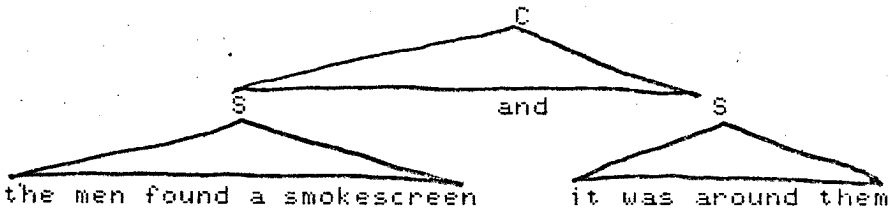
SC(PHI, the men)=true
AGR(PHI, the men)=true
RNR(PHI, the men)=true
AGR(PHIA, the men)=true
NEWCHAIN:the menC^PHIA

TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themA	a smokescreenB^PHIA
the menC^PHIA	

-PHI-	-them-
PHIA	themA

(10.8) The men found a smokescreen and it was around them.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
the men	-	-	-	+	+	-	+	-
a smokescr	-	-	-	+	-	?	-	-
it	+	-	-	+	-	?	-	-
them	+	-	-	+	+	?	?	-

INITTABLE

TABLE

-the men- the menA	-a smokescreen- a smokescreenA
-it- itA	-them- themA

SC(them, it)=true
AGR(them, it)=false

SC(them, a smokescreen)=true
AGR(them, a smokescreen)=false

SC(them, the men)=true
AGR(them, the men)=true
RNR(them, the men)=true
AGR(themA, the men)=true
NEWCHAIN:the menB^themA

TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themA	

-it-	-them-
itA	themA

SC(it, them)=false

SC(it, a smokescreen)=true
AGR(it, a smokescreen)=true
RNR(it, a smokescreen)=true
AGR(itA, a smokescreen)=true
NEWCHAIN:a smokescreenB^itA

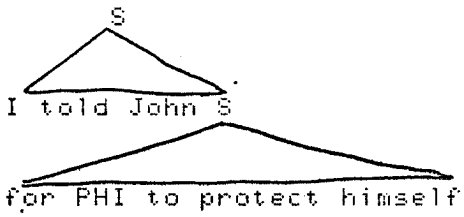
TABLE

-the men-	-a smokescreen-
the menA	a smokescreenA
the menB^themA	a smokescreenB^itA

-it-	-them-
itA	themA

SC(it, the men)=true
AGR(it, the men)=false

(10.9) I told John to protect himself.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
I	+	+	-	-	-	?	+	-
John	-	-	-	+	-	-	+	-
PHI	+	?	?	?	?	?	?	-
himself	+	-	-	+	-	-	+	+

INITTABLE

TABLE

-I- IA	-John- JohnA
-PHI- PHIA	-himself- himselfA

SC(himself, PHI)=true
AGR(himself, PHI)=true
RNR(himself, PHI)=true
AGR(himselfA, PHI)=true
NEWCHAIN:PHIB^himselfA

TABLE

-I-	-John-
IA	JohnA

-PHI-	-himself-
PHIA	himselfA
PHIB^himselfA	

SC(PHI, himself)=false

SC(PHI, John)=true
AGR(PHI, John)=true
RNR(PHI, John)=true
AGR(PHIA, John)=true
NEWCHAIN:JohnB^PHIA
AGR(PHIB^himselfA, John)=true
NEWCHAIN:JohnC^PHIB

TABLE

-I-	-John-
IA	JohnA
	JohnB^PHIA
	JohnC^PHIB

-PHI-	-himself-
PHIA	himselfA
PHIB^himselfA	

SC(PHI, I)=true
AGR(PHI, I)=true
RNR(PHI, I)=true
AGR(PHIA, I)=true
NEWCHAIN:IB^PHIA
AGR(PHIB^himselfA, I)=false

TABLE

-I-	-John-
IA	JohnA
IB^PHIA	JohnB^PHIA
	JohnC^PHIB

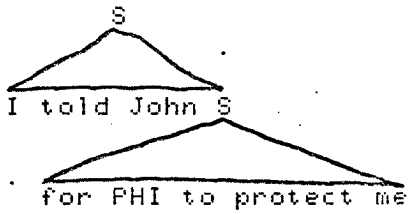
-PHI-	-himself-
PHIA	himselfA
PHIB^himselfA	

SC(I, himself)=false

SC(I, PHI)=false

SC(I, John)=false

(10.10) I told John to protect me.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
I	+	+	-	-	-	?	+	-
John	-	-	-	+	-	-	+	-
PHI	+	?	?	?	?	?	?	-
me	+	+	-	-	-	?	+	-

INITTABLE

TABLE

-I-	-John-
IA	JohnA
-PHI-	-me-
PHIA	meA

SC(me, PHI)=true
AGR(me, PHI)=true
RNR(me, PHI)=false

SC(me, John)=true
AGR(me, John)=false

SC(me, I)=true
AGR(me, I)=true
RNR(me, I)=true
AGR(meA, I)=true
NEWCHAIN:IB^meA

TABLE

-I-	-John-
IA	JohnA
IB^meA	

-PHI-	-me-
PHIA	meA

SC(PHI, me)=false

SC(PHI, John)=true
AGR(PHI, John)=true
RNR(PHI, John)=true
AGR(PHIA, John)=true
NEWCHAIN:JohnB^PHIA

TABLE

-I-	-John-
IA	JohnA
IB^meA	JohnB^PHIA

-PHI-	-me-
PHIA	meA

SC(PHI, I)=true
AGR(PHI, I)=true
RNR(PHI, I)=true
AGR(PHIA, I)=true
NEWCHAIN:IC^PHIA

TABLE

-I-	-John-
IA	JohnA
IB^meA	JohnB^PHIA
IC^PHIA	

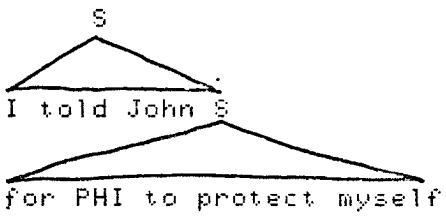
-PHI-	-me-
PHIA	meA

SC(I, me)=false

SC(I, PHI)=false

SC(I, John)=false

(10.11) I told John to protect myself.



FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
I	+	+	-	-	-	?	+	-
John	-	-	-	+	-	-	+	-
PHI	+	?	?	?	?	?	?	-
myself	+	+	-	-	-	?	+	+

INITTABLE

TABLE

-I-	-John-
IA	JohnA
-PHI-	-myself-
PHIA	myselfA

SC(myself, PHI)=true
AGR(myself, PHI)=true
RNR(myself, PHI)=true
AGR(myselfA, PHI)=true
NEWCHAIN:PHIB^myselfA

TABLE

-I-	-John-
IA	JohnA
-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

SC(PHI, myself)=false

SC(PHI, John)=true
AGR(PHI, John)=true
RNR(PHI, John)=true
AGR(PHIA, John)=true
NEWCHAIN:JohnB^PHIA
AGR(PHIB^myselfA, John)=false

TABLE

-I-	-John-
IA	JohnA
	JohnB^PHIA

-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

SC(PHI, I)=true
AGR(PHI, I)=true
RNR(PHI, I)=true
AGR(PHIA, I)=true
NEWCHAIN:IB^PHIA
AGR(PHIB^myselfA, I)=true
NEWCHAIN:IC^PHIB

TABLE

-I-	-John-
IA	JohnA
IB^PHIA	JohnB^PHIA
IC^PHIB	

-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

SC(I, myself)=false

SC(I, PHI)=false

SC(I, John)=false

Chapter 11. Table Interpretation.

The Table Interpreter module defines function Interpret and has the form shown in Figure 11.1.

```
"      mocule table_interpreter;
"          use globals, node_processor;
"          define function interpret(systable:node_pointer)
"                                                  :systrees;
"          implement
"              (*$I TABLE_INTR*)
"          end;
"
"
```

Figure 11.1. Skeleton Of The Table Interpreter.

Basically, after the chaining table is created, a number of chains are implicitly defined by the chaining table anc it is the job of the Table Interpreter to mesh these chains back into copies of the system tree, returning all trees defined by legitimate interpretations.

A ncnpronomial E-node with the E-nodes that are traced by walkirg down chainlinks until a nil chainlink is reached constitute a chain. A set of chains defined by the chaining table which cover all the pronomial N-nodes and do not intersect constitute a legitimate interpretation.

Take the table given in Figure 11.2 as an example.

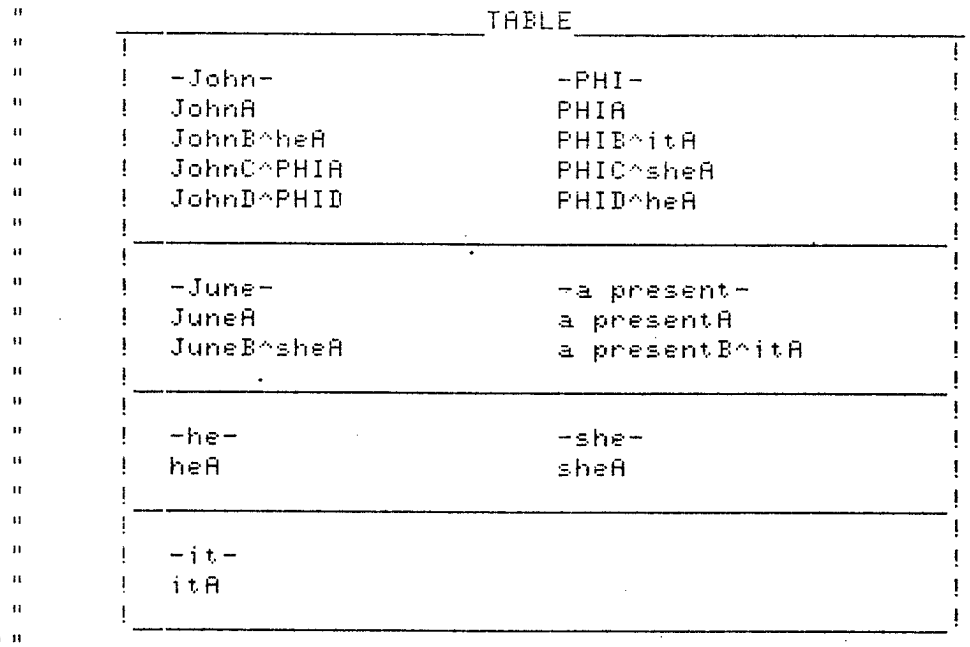


Figure 11.2. Typical Chaining Table.

The chains present in Figure 11.2 are exactly (11.3)-(11.10) given below. Note that no chain begins with a pronoun. (Here we are staying with the convention of Chapter 10 where a "^" symbol indicates a chainlink).

(11.3) JohnA
(11.4) JohnB^heA
(11.5) JohnC^PHIA
(11.6) JohnD^PHID^heA
(11.7) JuneA
(11.8) JuneB^sheA
(11.9) a presentA
(11.10) a presentB^itA

The only interpretation derivable from Figure 11.2 is (11.11).

(11.11) JohnD^PHID^heA JuneB^sheA a presentB^itA

Exactly how chaining information from a table to a

system parse tree will have to be system dependent, but we can imagine that noun phrases in a system parse tree are some kinc of list elements which have linked to them, among other things, lists corresponding to their semantics. It is up to the table interpreter to set any chainlinks inside the semantics of the noun phrases of the system parse tree. The Semantic Processor module of the system should then be powerful enough to be able to handle the kind of coordination that chainlinks imply.

This strategy has a number of possibilities that simple methods cf coreferencing are just not able to handle. Consider sentence (11.12) for example.

(11.12) Jack's house burned down, but he rebuilt it.

We can't really say that it corefers with Jack's house as Jack's house is some object that existed in the past and has stopped existing while it refers to some new object. This does not mean that it cannot chain from Jack's house, however, and indeed it should. The information the Semantic Processor Module needs to give meaning to it is contained in Jack's hcuse, and so there must be a chainlink from Jack's house to it in order for the Semantic Processor to give meaning to it.

A similar result holds for quantifiers. We see that (11.13) is not equivalent to (11.14).

(11.13) Every connosieur loves his wine and cheese.<>

(11.14) Every connosieur loves every connosieur's wine

and cheese.

Quite clearly, his cannot be replaced by every connosieur and preserve the meaning of the sentence. Instead, (11.13) has more the meaning given by (11.15).

(11.15) (For all x: x is a connosieur)(x loves x's wine and cheese.)

A number of other examples are pointed out by Bresnan [3].

(11.16) All Italians think they are handsome.<>
(11.17) All Italians think all Italians are handsome.
(11.18) Every Italian thinks he is handsome.<>
(11.19) Every Italian thinks every Italian is handsome.
(11.20) Any Italian would die for his mother.<>
(11.21) Any Italian would die for and Italian's mother.
(11.22) Every Italian thinks that he alone is handsome.<>
(11.23) *Every Italian thinks that every Italian alone is handsome.
(11.24) One girl claimed that she herself could read Homer.<>
(11.25) *One girl claimed that one girl herself could read Homer.

It appears that the proper interpretation for a pronoun chained to a quantified noun phrase within the scope of quantification is for the pronoun to act as a bound variable.

Wher the pronoun is outside the scope of quantification, it is a different story. Consider (11.26) and (11.27) from Evans [6].

(11.26) John owns some sheep and Harry vaccinates them.
(11.27) Mary danced with many boys and they found her interesting.

This time the pronouns are chaining to quantified noun phrases, but do not themselves lie within the scope of quantification. Instead, they appear to refer to the range of the quantification.

Similar results hold for (11.28)-(11.31) from Sidner [29].

(11.28) John lost a pen yesterday and Bill found one today.
(11.29) John claimed to have found the solution to the problem, but Bill was sure he had found it.
(11.30) John wants to catch a fish and eat it for supper.
(11.31) No one would put the blame on himself.

The problems mentioned above are all rather tricky, but viewing them from the vantage point of chaining sheds more light on them than viewing them through some kind of coreferernce. The moral of the story seems to be that anaphora is not coreference.

Usirg the table interpreter now, we present some more examples.

(11.32) Sue told Sandy about herself.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Sue	-	-	-	+	-	+	+	-
Sandy	-	-	-	+	-	+	+	-
herself	+	-	-	+	-	+	+	+

INITTABLE

TABLE

-Sue-	-Sandy-
SueA	SandyA

-herself-
herselfA

SC(herself, Sandy)=true
AGR(herself, Sandy)=true
RNR(herself, Sandy)=true
AGR(herselfA, Sandy)=true
NEWCHAIN:SandyB^herselfA

TABLE

-Sue-	-Sandy-
SueA	SandyA
	SandyB^herselfA

-herself-
herselfA

SC(herself, Sue)=true
AGR(herself, Sue)=true
RNR(herself, Sue)=true
AGR(herselfA, Sue)=true
NEWCHAIN:SueB^herselfA

TABLE

-Sue-	-Sandy-
SueA	SandyA
SueB^herselfA	SandyB^herselfA

-herself-
herselfA

INTERPRETATIONS

SandyB^herselfA
SueB^herselfA

(11.33) *Jill kept talking about himself.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Jill	-	-	-	+	-	+	+	-
himself	+	-	-	+	-	-	+	+

INITTABLE

TABLE

-Jill-	-himself-
JillA	himselfA

SC(himself, Jill)=true
AGR(himself, Jill)=false

INTERPRETATIONS

NONE

(11.34) Does Jack's making a pig of himself bother Bill?

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Jack	-	-	-	+	-	-	+	-
a pig	-	-	-	+	-	?	-	-
himself	+	-	-	+	-	-	+	+
Bill	-	-	-	+	-	-	+	-

INITTABLE

TABLE

-Jack-	-a pig-
JackA	a pigA
-himself-	-Bill-
himselfA	BillA

SC(himself, a pig)=true
AGR(himself, a pig)=false

SC(himself, Jack)=true
AGR(himself, Jack)=true
RNR(himself, Jack)=true
AGR(himselfA, Jack)=true
NEWCHAIN:JackB^himselfA

TABLE

-Jack-	-a pig-
JackA	a pigA
JackB^himselfA	
-himself-	-Bill-
himselfA	BillA

INTERPRETATIONS

JackB^himselfA

(11.34) John wants to give June a present, but he is afraid she won't like it.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
John	-	-	-	+	-	-	+	-
PHI	+	?	?	?	?	?	?	-
June	-	-	-	+	-	+	+	-
a present	-	-	-	+	-	?	-	-
he	+	-	-	+	-	-	+	-
she	+	-	-	+	-	+	+	-
it	-	-	-	+	-	?	-	-

INITTABLE

TABLE

-John- JohnA	-PHI- PHIA
-June- JuneA	-a present- a presentA
-he- heA	-she- sheA
-it- itA	

SC(she, it)=false

SC(she, he)=true
AGR(she, he)=false

SC(she, a present)=true
AGR(she, a present)=false

SC(she, June)=true
AGR(she, June)=true
RNR(she, June)=true
AGR(sheA, June)=true
NEWCHAIN:JuneB^sheA

TABLE

-John- JohnA	-PHI- PHIA
-June- JuneA JuneB^sheA	-a present- a presentA
-he- heA	-she- sheA
-it- itA	

SC(she, PHI)=true
AGR(she, PHI)=true
RNR(she, PHI)=true
AGR(sheA, PHI)=true
NEWCHAIN:PHIB^sheA

TABLE

-John- JohnA	-PHI- PHIA PHIB^sheA
-June- JuneA JuneB^sheA	-a present- a presentA
-he- heA	-she- sheA
-it- itA	

SC(she, John)=true
AGR(she, John)=false

SC(he, it)=false

SC(he, she)=false

SC(he, a present)=true
AGR(he, a present)=false

SC(he, June)=true
AGR(he, June)=false

SC(he, PHI)=true
AGR(he, PHI)=true
RNR(he, PHI)=true
AGR(heA, PHI)=true
NEWCHAIN:PHIC^heA

TABLE

-John-	-PHI-
JohnA	PHIA
	PHIB^sheA
	PHIC^heA
-June-	-a present-
JuneA	a presentA
JuneB^sheA	
-he-	-she-
heA	sheA
-it-	
itA	

SC(he, John)=true
AGR(he, John)=true
RNR(he, John)=true
AGR(heA, John)=true
NEWCHAIN:JohnB^heA

TABLE

-John-	-PHI-
JohnA	PHIA
JohnB^heA	PHIB^sheA
	PHIC^heA
-June-	-a present-
JuneA	a presentA
JuneB^sheA	
-he-	-she-
heA	sheA
-it-	
itA	

SC(PHI, it)=false

SC(PHI, she)=false

SC(PHI, he)=false

SC(PHI, a present)=false

SC(PHI, June)=false

SC(PHI, John)=true
AGR(PHI, John)=true
RNR(PHI, John)=true
AGR(PHIA, John)=true
NEWCHAIN:JohnC^PHIA
AGR(PHIB^sheA, John)=false
AGR(PHIC^heA, John)=true
NEWCHAIN:JohnD^PHIC

TABLE

-John-	-PHI-
JohnA	PHIA
JohnB^heA	PHIB^sheA
JohnC^PHIA	PHIC^heA
JohnD^PHIC	

-June-	-a present-
JuneA	a presentA
JuneB^sheA	

-he-	-she-
heA	sheA

-it-
itA

INTERPRETATIONS

JohnD^PHIC^heA	JuneB^sheA

(11.35) Ernie doesn't like Bernie, because he is such an asshole.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf
Ernie	-	-	-	+	-	-	+	-
Bernie	-	-	-	+	-	-	+	-
he	+	-	-	+	-	-	+	-
such an as	-	-	-	+	-	?	+	-

INITTABLE

TABLE

-Ernie- ErnieA	-Bernie- BernieA
-he- heA	-such an asshole- such an assholeA

SC(he, such an asshole)=false

SC(he, Bernie)=true
AGR(he, Bernie)=true
RNR(he, Bernie)=true
AGR(heA, Bernie)=true
NEWCHAIN:BernieB^heA

TABLE

-Ernie- ErnieA	-Bernie- BernieA BernieB^heA
-he- heA	-such an asshole- such an assholeA

SC(he, Ernie)=true
AGR(he, Ernie)=true
RNR(he, Ernie)=true
AGR(heA, Ernie)=true
NEWCHAIN:ErnieB^heA

TABLE

-Ernie-	-Bernie-
ErnieA	BernieA
ErnieB^heA	BernieB^heA
-he-	-such an asshole-
heA	such an assholeA

Chapter 12. Genitives.

Very little modification to what has been said so far is necessary to implement attributive possessive pronouns. Recall that the attributive possessive pronouns are those pronouns listed in (12.1).

(12.1) my, our, your, her, his, its, their

Examining sentences like (12.2)-(12.5) reveals that reflexive pronouns don't chain to genitives within the same simplex. On the other hand, nonreflexive pronouns can.

(12.2) Mary's father killed himself.
(12.3) *Mary's father killed him.
(12.4) *Mary's father killed herself.
(12.5) Mary's father killed her.

The same conclusions also hold for of-genitives. Compare sentences (12.6)-(12.9) to (12.2)-(12.5).

(12.6) The father of Mary killed himself.
(12.7) *The father of Mary killed him.
(12.8) *The father of Mary killed herself.
(12.9) The father of Mary killed her.

The easiest way to handle genitives, apparently, is to introduce a new feature, gen, for genitive and to modify the Reflexive Nonreflexive Rule to handle genitives. The new form of the Reflexive Nonreflexive Rule is shown below in Figure 12.10.

```
function rnr(n1,n2:node_pointer):boolean;
(Reflexive Nonreflexive Rule)
    var ftr1,ftr2:features;
begin
    n1:=n1^.nplink;
    n2:=n2^.nplink;
    ftr1:=n1^.ftr;
    ftr2:=n2^.ftr;
    if ftr2[gen]=plus then rnr:=true
    else case ftr1[rpf] of
        plus:     rnr:=(n1^.uplink=n2^.uplink)
                       and (ftr2[gen]<>plus);
        question: (doesn't occur);
        minus:    rnr:=(n1^.uplink<>n2^.uplink)
                       or (ftr2[gen]=plus);
    end;
end;
```

Figure 12.10. New Reflexive Nonreflexive Rule.

The examples following illustrate the interpretation of attributive possessive pronouns and pronouns in the context of genitives.

(12.11) Mary's mother cooks only for herself.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf	gen
Mary	-	-	-	+	-	+	+	-	+
mother	-	-	-	+	-	+	+	-	-
herself	+	-	-	+	-	+	+	+	-

INITTABLE

TABLE

-Mary-	-mother-
maryA	motherA
-herself-	
herselfA	

SC(herself, mother)=true
AGR(herself, mother)=true
RNR(herself, mother)=true
AGR(herselfA, mother)=true
NEWCHAIN:motherB^herselfA

TABLE

-Mary-	-mother-
maryA	motherA
	motherB^herselfA
-herself-	
herselfA	

SC(herself, Mary)=true
AGR(herself, Mary)=true
RNR(herself, Mary)=false

INTERPRETATIONS

motherB^herselfA

(12.12) Mary's mother cooks only for her.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf	gen
Mary	-	-	-	+	-	+	+	-	+
mother	-	-	-	+	-	+	+	-	-
her	+	-	-	+	-	+	+	-	-

INITTABLE

TABLE

-Mary-	-mother-
MaryA	motherA

-her-
herA

SC(her, mother)=true
AGR(her, mother)=true
RNR(her, mother)=false

SC(her, Mary)=true
AGR(her, Mary)=true
RNR(her, Mary)=true
AGR(herA, Mary)=true
NEWCHAIN:MaryB^herA

TABLE

-Mary-	-mother-
MaryA	motherA
MaryB^herA	

-her-
herA

INTERPRETATIONS

MaryB^herA

(12.13) Mary's mother cooks only for her mother.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf	gen
Mary	-	-	-	+	-	+	+	-	+
mother1	-	-	-	+	-	+	+	-	-
her	+	-	-	+	-	+	+	-	+
mother2	-	-	-	+	-	+	+	-	-

INITTABLE

TABLE

-Mary- MaryA	-mother1- mother1A
-her- herA	-mother2- mother2A

SC(her, mother2)=false

SC(her, mother1)=true
AGR(her, mother1)=true
RNR(her, mother1)=true
AGR(herA, mother1)=true
NEWCHAIN:mother1B^herA

TABLE

-Mary- MaryA	-mother1- mother1A mother1B^herA
-her- herA	-mother2- mother2A

SC(her, Mary)=true
AGR(her, Mary)=true
RNR(her, Mary)=true
AGR(herA, Mary)=true
NEWCHAIN:MaryB^herA

TABLE

-Mary-	-mother1-
MaryA	mother1A
MaryB^herA	mother1B^herA
-her-	-mother2-
herA	mother2A

INTERPRETATIONS

mother1B^herA
MaryB^herA

Chapter 13. Focussing.

Extrasentential anaphora and ellipsis is possible through the maintenance of a focus of conversation. This maintenance is known as focussing and has been described at length by Grosz [18] and Sidner [17]. By focus of a conversation, we mwan the common view of the participants of a converation of what their conversation is about. Focussing is useful because it allows the participants of a conversation to avoid redundant repetition of old material. Assuming focussing is desirable in a computer natural language system, how do we implement it?

Grosz has examined task dialogues in which an expert helps an apprentice to assemble a mechanical air compressor. She finds it convenient to represent the focus of conversation as a set of overlapping focus spaces, where each focus space is a colection of objects. One focus space is active and the others are open. When a focus space is no longer needed, it is closed. One of Grosz's assumptions is that goals and subgoals are definable and recognizable in a task dialogue system with the consequence that in any conversation there is an open focus space hiarchy with the active focus space at the bottom of the hiarchy.

Sidner has approached the broblem of focussing from a different perspective by analyzing monologues. For Sidner,

focus is kept track of through a discourse focus, actor focus, potential discourse foci, potential actor foci, discourse focus stack, and actor focus stack. Sidner's work, which came after Grosz's, is very commendable for the algorithms she presents, although most of these are fairly sketchy.

In our approach, we will treat the focus of conversation as a collection of nonpronomial N-nodes. Among the N-nodes that we would ordinairly expect to always be in focus are the I and you of a conversation. To get a handle on the focussed N-nodes, we dominate them by an S-node just as if they all had occurred in one simplex. So, for example, if I0 and you0 are the nonpronomial N-nodes currently in focus, then the current focus representation is given by a structure like Figure 13.1.

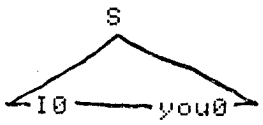


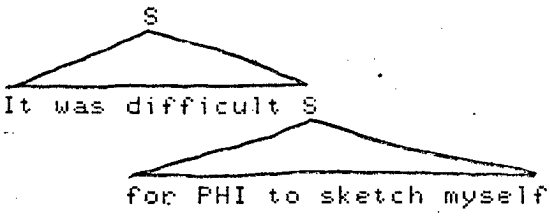
Figure 13.1. Typical Focus Representation.

When it comes time to analyze a sentence, the current focus representation is attached to the C-S-N parse tree of the sentence via a C-node which dominates them both. This makes the focussed N-nodes available to the N-nodes of the C-S-N parse tree for chaining.

As an example, suppose I0 and you0 are in focus and the

current input sentence is (13.2) from Grinder [8].

(13.2) It was difficult to sketch myself.



The C-S-N parse tree of (13.2) will have a form like that indicated below in Figure 13.3.

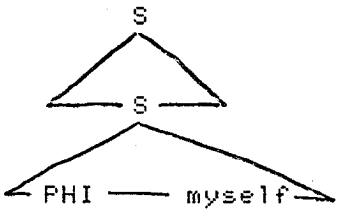


Figure 13.3. C-S-N Parse Tree of (13.2).

As the parse tree now stands in Fiugre 2, myself may chain to PHI, but PHI does not have an N-node to chain to. Thus, there are no legitimate interpretations without focussing. With focussing, the parser attaches the current focus representation conataining I0 and you0 to the C-S-N parse tree by a C-node obtaining the new C-S-N parse tree shown in Figure 13.4.

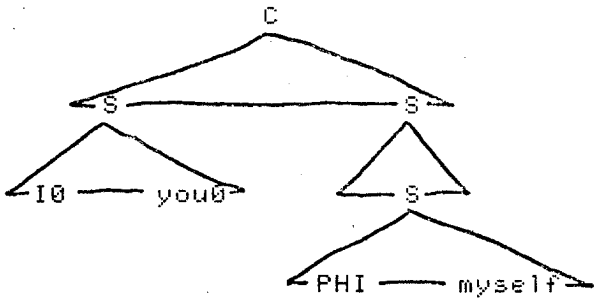


Figure 13:4. C-S-N Parse Tree With Focussing.

Now myself can chain from PHI and PHI can chain from I0 giving us a legitimate interpretation of the C-S-N tree.

This kind of strategy exaplains a number of other examples from Grinder. Grinder lists (13.5)-(13.11) as grammatical.

(13.5) It was difficult for me to sketch myself.
(13.6) It was difficult for you to sketch yourself.
(13.7) It was difficult for him to sketch himself.
(13.8) It was difficult for her to sketch herself.
(13.9) It was difficult for us to sketch ourselves.
(13.10) It was difficult for you to sketch yourselves.
(13.11) It was difficult for them to sketch themselves.

After Equi-NP Deletion, Grinder lists only (13.12), (13.13), (13.16), and (13.17) as grammatical.

(13.5) It was difficult to sketch myself.
(13.6) It was difficult to sketch yourself.
(13.7) *It was difficult to sketch himself.
(13.8) *It was difficult to sketch herself.
(13.9) It was difficult to sketch ourselves.
(13.10) It was difficult to sketch yourselves.
(13.11) *It was difficult to sketch themselves.

The probable reaon this comes about is that we are used to thinking of I, you singular, us, and you plural as always being in focus while referents for he, she, and they are

ordinairly not in focus. Needless to say, if refernets for he she, or they are in focus, the situation shanges completely. This is shown by (13.12)-(13.14).

(13.12a) Nurse Bob Breezy gave up drawing.
(b) It was difficult to sketch himself.

(13.13a) Astronaut Linda Smith gave up drawing.
(b) It was difficult to sketch herself.

(13.14a) The bank embezzlers gave up drawing.
(b) It was difficult to sketch themselves.

To indicate that various N-nodes are in focus, we bracket them at the beginning of a sentence. Thus (13.15)-(13.17) are not interpretable while (13.18)-(13.20) are.

(13.15) *It was difficult to sketch himself.
(13.16) *It was difficult to sketch herself.
(13.17) *It was difficult to sketch themselves.
(13.18) [Bob Breezy] It was difficult to sketch himself.
(13.19) [Astronaut Linda Smith] It was difficult to sketch herself.
(13.20) [the bank embezzlers] It was difficult to sketch themselves.

The following examples involve resolution through focussing.

(13.21) [I0, you0] It was difficult to sketch myself.

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf	gen
I0	-	+	-	-	-	?	+	-	-
you0	-	-	+	-	-	?	+	-	-
PHI	+	?	?	?	?	?	?	-	-
myself	+	+	-	-	-	?	+	+	-

INITTABLE

TABLE

-I0-	-you0-
I0A	you0A
-PHI-	-myself-
PHIA	myselfA

SC(myself, PHI)=true
AGR(myself, PHI)=true
RNR(myself, PHI)=true
AGR(myselfA, PHI)=true
NEWCHAIN:PHIB^myselfA

TABLE

-I0-	-you0-
I0A	you0A
-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

SC(PHI, myself)=false

SC(PHI, you0)=true
AGR(PHI, you0)=true
RNR(PHI, you0)=true
AGR(PHIA, you0)=true
NEWCHAIN:you0B^PHIA
AGR(PHIB^myselfA, you0)=false

TABLE

-I0-	-you0-
I0A	you0A
	you0B^PHIA

-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

SC(PHI, I0)=true
AGR(PHI, I0)=true
RNR(PHI, I0)=true
AGR(PHIA, I0)=true
NEWCHAIN:I0B^PHIA
AGR(PHIB^myselfA, I0)=true
NEWCHAIN:I0C^PHIB

TABLE

-I0-	-you0-
I0A	you0A
I0B^PHIA	you0B^PHIA
I0C^PHIB	

-PHI-	-myself-
PHIA	myselfA
PHIB^myselfA	

INTERPRETATIONS

I0C^PHIB^myselfA

(13.22) Give me that!

FEATURES

	pnf	fpf	spf	tpf	plf	gnf	anf	rpf	gen
I0	-	+	-	-	-	?	+	-	-
you0	-	-	+	-	-	?	+	-	-
toy	-	-	-	+	-	?	-	-	-
PHI	+	?	?	?	?	?	?	-	-
me	+	+	-	-	-	?	+	-	-
that	+	-	-	+	-	?	-	-	-

INITTABLE

TABLE

-I0- I0A	-you0- you0A
-toy- toyA	-PHI- PHIA
-me- meA	-that- thatA

```
SC(that, me)=true
AGR(that, me)=false

SC(that, PHI)=true
AGR(that, PHI)=true
RNR(that, PHI)=false

SC(that, toy)=true
AGR(that, toy)=true
RNR(that, toy)=true
     AGR(thatA, toy)=true
     NEWCHAIN:toyB^thatA
```

TABLE

-I0- I0A	-you0- you0A
-toy- toyA toyB^thatA	-PHI- PHIA
-me- meA	-that- thatA

SC(that, you0)=true
AGR(that, you0)=false

SC(that, I0)=true
AGR(that, I0)=false

SC(me, that)=false

SC(me, PHI)=true
AGR(me, PHI)=true
RNR(me, PHI)=false

SC(me, toy)=true
AGR(me, toy)=false

SC(me, you0)=true
AGR(me, you0)=false

SC(me, I0)=true
AGR(me, I0)=true
RNR(me, I0)=true
AGR(meA, I0)=true
NEWCHAIN:I0B^meA

TABLE

-I0- I0A I0B^meA	-you0- you0A
-toy- toyA toyB^thatA	-PHI- PHIA
-me- meA	-that- thatA

SC(PHI, that)=false

SC(PHI, me)=false

SC(PHI, toy)=true
AGR(PHI, toy)=true
RNR(PHI, toy)=true
AGR(PHIA, toy)=true
NEWCHAIN:toyC^PHIA

TABLE

-I0-	-you0-
I0A	you0A
I0B^meA	
-toy-	-PHI-
toyA	PHIA
toyB^thatA	
toyC^PHIA	
-me-	-that-
meA	thatA

SC(PHI, you0)=true
AGR(PHI, you0)=true
RNR(PHI, you0)=true
AGR(PHIA, you0)=true
NEWCHAIN:you0B^PHIA

TABLE

-I0-	-you0-
I0A	you0A
I0B^meA	you0B^PHIA
-toy-	-PHI-
toyA	PHIA
toyB^thatA	
toyC^PHIA	
-me-	-that-
meA	thatA

SC(PHI, I0)=true
AGR(PHI, I0)=true
RNR(PHI, I0)=true
AGR(PHIA, I0)=true
NEWCHAIN:I0C^PHIA

TABLE

-I0-	-you0-
I0A	you0A
I0B^meA	you0B^PHIA
I0C^PHIA	
-toy-	-PHI-
toyA	PHIA
toyB^thatA	
toyC^PHIA	
-me-	-that-
meA	thatA

INTERPRETATIONS

I0B^meA	you0B^PHIA	toyB^thatA

Bibliography.

1. Bloom, David and Hayes, David G. "Designation in English." In Anaphora in Discourse, John Hinds, Ed., Linguistic Research, Inc., Edmonton, Alberta, Canada, 1978.

2. Bloomfield, Leonard. Language. Holt, Rinehart and Winston, Inc., New York, 1933.

3. Bresnan, Joan. "A Note on the Notion 'Idntity of Sense Anaphora.'" Linguistic Inquiry 2, 4, 1971, 589-596.

4. Chiba, Shuji. "A Note on Equi-NP Deletion." Linguistic Inquiry 2, 4, 1971, 539-540.

5. Chomsky, Noam. Syntactic Structures. Mouton & Co., Publishers, The Hague, The Netherlands, 1957.

6. Evans, Gareth. "Pronouns, Quantifiers, and Relative Clauses (I)." Canadian Journal of Philosophy 7, 3, 1977, 467-536.

7. Fauconnier, Gilles. "Do Quantifiers Branch?" Linguistic Inquiry 6, 4, 1975, 555-567.

8. Grinder, John. "Chains of Coreference." Linguistic Inquiry 2, 2, 1971 183-202.

9. Grosu, Alexander. "On the Nonunitary Nature of the Coordinate Structure Constraint." Linguistic Inquiry 4, 1, 1973, 88-92.

10. Grosz, Barbara J. The Representation and Use of Focus in Dialogue Understanding. Stanford Research

Institute, Techinical Note 151, Menlo Park, California, 1977.

11. Hankamer, Jorge and Sag, Ivan. "Deep and Surface Anaphora." Linguistic Inquiry 7, 3, 1976, 391-426.

12. Hockett, Charles F. A Course in Modern Linguistics. The Macmillan Company, New York, 1958.

13. Huddleston, Rodney. "A Survey of the Crossing Coreference Controversy Papers in Linguistics 11, 3-4, 1978, 295-319.

14. Jackendoff, Ray S. "Any Vs. Every." Linguistic Inquiry 3, 1, 1972, 119-120.

15. Jackendoff, Ray S. Semantic Interpretation in Generative Grammar. The MIT Press, Cambridge, Massachusetts, 1972.

16. Kuno, Susumu. "Lexical and Contextual Meaning." Linguistic Inquiry, 5, 3, 1974, 469-477.

17. Kuno, Susumu. "Pronominalization, Reflexivization, and Direct Discourse." Linguistic Inquiry 3, 2, 1972, 161-195.

18. Lakoff, George and Ross, John R. "A Note on Anaphoric Islands and Causatives." Linguistic Inquiry 3, 1, 1972, 121-125.

19. Langacker, Ronald W. "On Pronominalization and the Chain of Command." In Modern Studies in English, David A. Reible and Sanford A. Schane, Eds., Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.

20. Langendoen, D. Terence. Essentials of English Grammar. Holt, Rinehar and Winston, Inc., New York, 1970.

21. Lees, Robert B. The Grammar of English Nominalizations. Mouton & Co. Publishers, The Hague, The Netherlands, 1963.

22. Lees, Robert B. and Klima, Edward S. "Rules for English Pronominalization." Language 39, 1, 1936, 17-28.

23. Postal, Paul M. "On Coreferential Complement Subject Deletion." Linguistic Inquiry 1, 4, 1970, 439-500.

24. Postal, Paul M. "On So-Called 'Pronouns' in English." In The 19th Monograph on Language and Liguistics, F. Dinneen, Ed., Georgetown University Press, Washington, D.C., 1966.

25. Roberts, Paul. Modern Grammar. Harcourt, Brace & World, Inc., New York, 1967.

26. Ross, John R. "On the Cyclic Nature of English Pronominalization." In To Honor Roman Jakobson, III, The Hague, Mouton & Co., Printers, The Netherlands, 1967.

27. Sag, Ivan A. "The Nonunity of Anaphora." Linguistic Inquiry 10, 1, 1979, 152-164.

28. Saltarelli, Mario. "Focus on Focus: Propositional Generative Grammar. In Studies Presented to Robert B. Lees by His Students, Jerrold M. Sadock and Anthony L. Vanek, Eds., Linguistic Research, Inc., Edmonton, Alberta, Canada, 1970.

29. Sidner, Candace L. Towards a Computational Theory of Definite Anaphor Comprehension in English Discourse. MIT Artificial Intelligence Laboratory, Technical Report 537, Cambridge, Massachusetts, 1979.

30. Smith, Carlota S. "Ambiguos Sentences with And." In Modern Studies in English, David A. Reibel and Sanford A. Schane, Eds., . Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.

31. Quirk, Randolph and Greenbaum, Sidney. A Concise Grammar of Contempor English. Harcourt Brace Jovanovich, Inc., New York, 1973.

32. Yotsukura, Sayo. The Articles in English. Mouton & Co., Printers, The Hague, The Netherlands, 1970.

33. Wasow, Thomas. "Anaphoric Pronouns and Bound Variables." Language, 51, 2, 1975, 368-383.