American Journal of Computational Linguistics *Microfiche 29 : 25*

COMPUTER GENERATION OF SENTENCES BY SYSTEMIC GRAMMAR

JOHN SELF

Department of Information Science
University of Melbourne
Parkville, Victoria, Australia 3052

ABSTRACT

The paper describes a computer model of systemic grammar, a generative grammar for natural language. A program is explained which given the features of an item, determines the structure of that item according to a systemic grammar specified as data. The program thus demonstrates the principles of systemic grammar, a brief summary of the mechanics of which is also included. Some implications of the program for systemic grammar itself are discussed. In particular, it is shown that previous definitions of the operation of structure-building rules require modification.

1. Introduction

This paper describes a computer model of systemic grammar, a grammar for natural languages developed by Halliday and colleagues at University College, London (Halliday, 1961, 1970). Systemic grammar has recently been of interest to computational grammarians, primarily as a result of the impressive work of Winograd.(1972), who developed a natural language understanding system one component of which was strongly influenced by the principles of systemic grammar. More recently, Power (1974) has also investigated how systemic grammar can be used to analyse natural language. There have, however, been no attempts to use a computer to investigate systemic grammar itself. As Friedman (1971) says, in introducing her computer model of transformational grammar, adequate natural language grammars are bound to be so complex that some mechanical aid in investigating their properties will be mandatory.

The aims, then, of developing a computer model of systemic grammar are threefold. First, the model enables the grammar to be tested, i.e. it enables contradictions, ambiguities and incompletenesses in the grammar to be found. Secondly, the model enables systemic grammar itself to be improved, since the consequences of adjusting parameters and rules can be more easily followed. And, thirdly, the model serves as a demonstration of how systemic grammar 'works'

Earlier descriptions of systemic grammar were somewhat incomplete, but that of Hudson (1971) seems sufficiently precise to encourage the feeling that a computer program could be based upon it. The program described below generates (in the linguistic sense) natural language sentences, i.e. "assigns structural descriptions to sentences" (Chomsky, 1965). It is not concerned directly with understanding or producing sentences.

2. The Mechanics of Systemic Grammar

This section briefly describes the generative apparatus of systemic grammar - for a fuller discussion, and for linguistic justifications of the processes, the reader is referred to Hudson (1971), from which the example grammar and generations given later are taken.

In systemic grammar, "structures are entirely predictable from features: given all of an item's features, we can predict exactly what its structure will be" (Hudson, pg 87). In general terms, an item's features or classes are those categories to which it belongs irrespective of the particular sentence to which the item belongs; an item's functions are those categories to which it belongs as a result of its role in a sentence. For example,

"must" has the features MODAL-VERB, FINITE-VERB (among others) and in the sentence

"Must it grow darker?" has the functions !PRE-SUBJECT, !MOOD-FOCUS (among others)
(A preceding ! will be used to distinguish functions from features.) An item's structure is defined by its immediate constituents' functions and the sequence in which they occur.

Given all the features that an item has, the item's structure may be determined, according to systemic grammar, by the sequential application of rules of four kinds:

(1) feature-realisation rules

In the simplest case, these rules are of the form "if item has feature x then its structure will contain function y" - y is said to be the realisation of x. Some rules are conditional in that the realisation only holds if certain other features are or are not present. Also, some rules specify that two functions must be conflated, i.e. both functions apply to the same immediate constituent. (Further details of these and the following rules are given later when the program is discussed.) The application of the feature-realisation rules provides an unordered set of functions, some of which may be conflated.

(2) structure-building rules

These rules expand and order this set of functions to provide the structure of the item. Structure-building rules are themselves of four kinds, which in the simplest case are of the following forms

(a) addition rules: "if function y (or some combination of functions) is present, then so must be function z (possibly conflated with other functions)".

(b) conflation rules: "if some condition expressed in terms of functions is satisfied then some function must be conflated with certain other functions".

(c) sequence rules: "if two functions y and z are present then y must be conflated with, precede or not follow z".

(d) compatibility rules: "functions y and z must not be conflated".

Addition and conflation rules are only applicable if the resultant structure does not conflict with a sequence or compatibility rule. Structure-building rules are not extrinsically ordered in any way. After applying these rules, we have a complete specification of the item's structure, in that we have specified function-"bundles", each of which consists of the functions of one of the immediate constituents of the item.

(3) function-realisation rules

These rules specify which features are implied by an item's functions. They are of the form "if a structure contains function y the corresponding item must have feature x". Applied to the function-bundles obtained from (2), these rules help to determine the features possessed by the immediate constituents.

(4) systems

System networks specify which features are implied by other features. These networks are equivalent to rules of the form "if feature x is present then so is one (or all) of a set of features, and conversely". These rules expand a set of features (possibly the result of applying (3) , not necessarily into a complete set, since some features may be freely selected. The feature-realisation rules may then be recursively applied to this set of features, if required.

3. The Program

The program reads in a definition of a systemic grammar (provided as data so that it may be changed without necessitating major modifications to the program) and generates a structure from a specified list of features. The interested reader should have no difficulty in relating the rules given below to the grammar given by Hudson (pg. 53-101). The rules are shown in the form in which they are presented to the program, and are numbered to ease explanation and understanding of the program's execution. In order to enable the reader to follow the computer generations given later, an English interpretation of selected rules follows:

(1) feature-realisation rules

Rule 13 (below) means "if an item has the feature INTERROGATIVE then, provided it also is DEPENDENT, its structure contains the function !QUESTION (which is thereby introduced if not already present) conflated with !BINDER". Similarly, rule 32

NO	FEATURE	REALISATION	CONDITION
1	(CLAUSE	!PROCESS)	
2	(PHRASE	!HEAD)	
3	(WORD	!STEM)	
4	(INDEPENDENT	-)	
5	(DEPENDENT	!BINDER)	
6	(DEPENDENT	!SUBJECT)	
7	(DEPENDENT	!FINITE)	
8	(IMPERATIVE	-)	
9	(INDICATIVE	!SUBJECT)	
10	(INDICATIVE	!FINITE)	
11	(DECLARATIVE	-)	
12	(INTERROGATIVE	!MOOD-FOCUS	INDEPENDENT)
13	(INTERROGATIVE	(+ !QUESTION = !BINDER) DEPENDENT)	
14	(POLAR	-)	
15	(NON-POLAR	-)	
16	(WH	(+ !QUESTION = !MOOD-FOCUS) INDEPENDENT)	
17	(ALTERNATIVE	!ALTERNATIVE)	
18	(SUBJECT-FOCUS	(!QUESTION = !SUBJECT) WH)	
19	(SUBJECT-FOCUS	(!ALTERNATIVE = !SUBJECT) ALTERNATIVE)	
20	(MODAL	!MODAL)	
21	(NON-SUBJECT-FOCUS	-)	
22	(NON-MODAL	-)	
23	(INTRANSITIVE	-)	
24	(TRANSITIVE	!GOAL)	
25	(TRANSITIVE	!ACTOR	(NOT ACTOR-UNSPECIFIED))
26	(TRANSITIVE	!TRANSITIVE)	
27	(ATTRIBUTIVE	(+ !ATTRIBUANT = !SUBJECT))	
28	(ATTRIBUTIVE	!ATTRIBUTE)	
29	(ATTRIBUTIVE	!COPULAR)	
30	(NON-ATTRIBUTIVE	(+ !ACTOR = !SUBJECT))	
31	(NON-ATTRIBUTIVE	!INTRANS)	
32	(ACTIVE	(!ACTOR = !SUBJECT))	
33	(PASSIVE	(!GOAL = !SUBJECT))	
34	(PASSIVE	!PASSIVE)	
35	(ACTOR-SPECIFIED	(+ !AGENT = !ACTOR))	
36	(ACTOR-UNSPECIFIED	-)	

may conflate !ACTOR and !SUBJECT, but only if both are already present. (There are complications, explained later, when the function to be conflated with, e.g. !BINDER and !SUBJECT above, is absent.)

(2) structure-building rules

(a) Addition rule 2 means "!MODAL and !PASSIVE, if present, must be conflated with !POST-SUBJECT, added if necessary". Rule 1 means "if !MOOD-FOCUS is present but not conflated with !SUBJECT then !PRE-SUBJECT must be added if not already pre t"

NO	ADDITION RULE	CONDITION
1	((+ !PRE-SUBJECT)	(+ !MOOD-FOCUS # !SUBJECT))
2	((+ !POST-SUBJECT = !MODAL !PASSIVE))	
3	((+ !POST-VERB = !ATTRIBUTE !ACTOR !GOAL))	
4	((+ !EN = !PROCESS)	(+ !PASSIVE))

(b) Conflation rule 1 means "if !MOOD-FOCUS is not conflated with !QUESTION then !PRE-SUBJECT and !MOOD-FOCUS must be conflated, if present".

NO	CONFLATION RULE	CONDITION
1	((!PRE-SUBJECT = !MOOD-FOCUS)	(!MOOD-FOCUS # !QUESTION))
2	((!PROCESS = !COPULAR !TRANSITIVE !INTRANS))	

(c) Sequence rule 1 means "whichever of !MOOD-FOCUS or !BINDER is present, if either, will precede or be conflated with the first of !PRE-SUBJECT and !SUBJECT, if present, which, if both are present, will be in the specified order, and !POST-SUBJECT, if present, will follow the last of these functions, if any, and !PROCESS, if present, will follow or be conflated with the last of these functions if any, and !POST-VERB, if present, will follow the last of these functions, if any".

```
NO  SEQUENCE RULE

1  ((!MOOD-FOCUS OR !BINDER)
     => (!PRE-SUBJECT -> !SUBJECT)
     -> !POST-SUBJECT
     => !PROCESS
     -> !POST-VERB)
2  (!FINITE
     = (!PRE-SUBJECT
     = (!MODAL -> !PASSIVE)
     -> !PROCESS))
```

(d) Compatibility rule 1 means "!POST-SUBJECT must not be conflated with !PRE-SUBJECT"

```
NO  COMPATIBILITY RULE

1  ((!POST-SUBJECT # !PRE-SUBJECT))
2  ((!POST-VERB # !QUESTION))
3  ((!POST-VERB # !BINDER))
4  ((!POST-VERB # !SUBJECT))
```

(3) function-realisation rules

Rule 12 means "if an item has none of the functions !SUBJECT, !GOAL, !ATTRIBUTE or !AGENT then if it has !BINDER, it has the feature CONJUNCTION"

NO	FUNCTION	REALISATION	CONDITION
1	(!COPULAR	COPULAR-VERB)	
2	(!EN	EN-FORM)	
3	(!FINITE	FINITE-VERB)	
4	(!INTRANS	INTRANSITIVE-VERB)	
5	(!MODAL	MODAL-VERB)	
6	(!PASSIVE	BE)	
7	(!PROCESS	LEXICAL-VERB)	
8	(!TRANSITIVE	TRANSITIVE-VERB)	
9	(!AGENT	PREPOSITIONAL)	
10	(!ALTERNATIVE	DISJUNCTIVE)	
11	(!ATTRIBUTIVE	(OR ADJECTIVAL NOMINAL PREPOSITIONAL))	
12	(!BINDER	CONJUNCTION	(NOT (OR !SUBJECT !GOAL !ATTRIBUTE !AGENT)))
13	(!GOAL	(OR NOMINAL DEPENDENT))	
14	(!QUESTION	QUESTIONING)	
15	(!SUBJECT	(OR NOMINAL DEPENDENT))	

(4) systems

Rule 10* means "an item with feature INTRANSITIVE also has one of the features ATTRIBUTIVE and NON-ATTRIBUTIVE, and also has the features naming its supersystems, i.e. 9, 23 and 1, i.e. CLAUSE and ITEM". The *OR in the subsystems column indicates

NO	NAME (IF ANY)	SUPERSYSTEM	SUBSYSTEMS
1	(ITEM	-	(*OR 23 24 15))
23	(CLAUSE	1	(AND 2 9))
2	(-	23	(*OR 3 25))
3	(INDEPENDENT	2	(OR IMPERATIVE 26))
25	(DEPENDENT	2	27)
26	(INDICATIVE	3	27)
27	(-	(OR 25 26)	(AND 4 8))
4	(-	27	(OR DECLARATIVE 5))
5	(INTERROGATIVE	4	(OR POLAR 28))
28	(NON-POLAR	5	(AND 6 7))
6	(-	28	(OR WH ALTERNATIVE))
7	(-	28	(OR SUBJECT-FOCUS NON-SUBJECT-FOCUS))
8	(-	27	(OR MODAL NON-MODAL))
9	(-	23	(OR 10 11))
10	(INTRANSITIVE	9	(OR ATTRIBUTIVE NON-ATTRIBUTIVE))
11	(TRANSITIVE	9	(OR ACTIVE 12))
12	(PASSIVE	11	(OR ACTOR-SPECIFIED ACTOR-UNSPECIFIED))
24	(PHRASE	1	(AND 13 14))
13	(-	24	(OR NOMINAL ADJECTIVAL ADVERBIAL PREPOSITIONAL))
14	(-	24	(*OR NON-QUESTIONING QUESTIONING))
15	(WORD	1	(OR 29 CONJUNCTION))
29	(VERB	15	(AND 16 19))
16	(-	29	(*OR 17 18))
17	(NON-FINITE-VERB	16	(*OR FORM-0 EN-FORM ING-FORM))
18	(FINITE-VERB	16	(OR PAST-VERB PRESENT-VERB))
19	(-	29	(*OR 20 22))
20	(GRAMMATICAL-VERB	19	(*OR 21 MODAL-VERB))
21	(NON-MODAL-VERB	20	(*OR DO BE HAVE))
22	(LEXICAL-VERB	19	(OR COPULAR-VERB INTRANSITIVE-VERB TRANSITIVE-VERB))

* the systems are so numbered to correspond with Hudson's labellings (pg. 71).

that the first named feature or rule is the "default" option, taken unless there are environmental reasons for selecting another.

The rules of the grammar are in fact input and stored in the form of McCarthy lists (McCarthy, 1965), and the program is written in a list-processing extension of BCPL (Self, 1975).

It is important to realise that the rules are not extrinsically ordered in any way, and that the program may (conceptually) execute the rules in any order, with the objective of finding a structure consistent with all rules. Hence, rules are executed recursively, with backtracking when inconsistenices become apparent.

The generation of the structure of a sentence with the features CLAUSE, INDEPENDENT, INDICATIVE, INTERROGATIVE, NON-POLAR, WH, SUBJECT-FOCUS, NON-MODAL, TRANSITIVE, PASSIVE and ACTOR-UNSPECIFIED, e.g. "Which of the tents were errected?" (Hudson, pg. 100), is shown below. Each piece of output is preceded by an indication of the rule that has been executed, e.g. FR 1 indicates the first feature-realisation rule. In the printout of structures,

A
B

indicates that A, B, .. (which may be functions or structures) are conflated. Similarly,

```
->          =>          ?
  A           A           A
  B           B           B
```

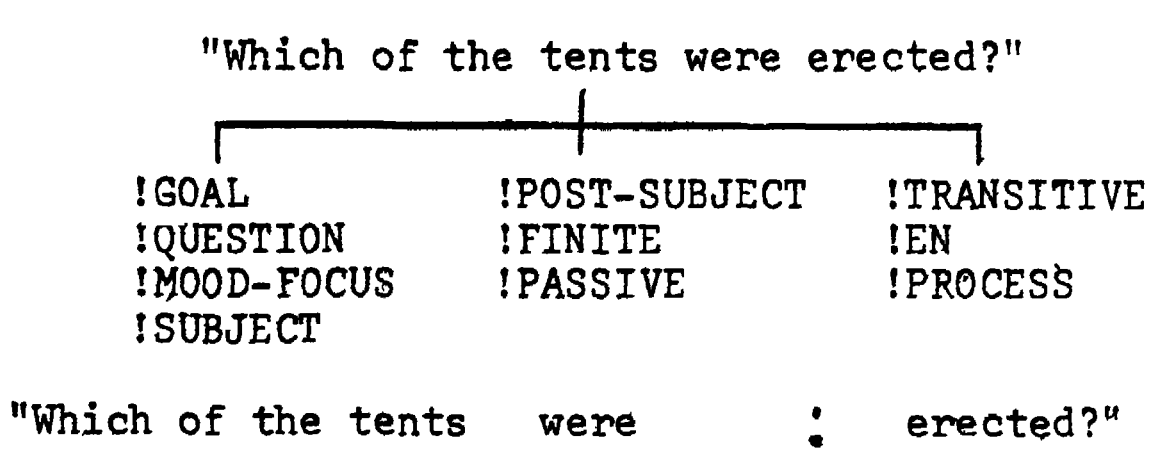
indicate, respectively, that A precedes B, that A precedes or is conflated with B, and that the order of A and B is undetermined. When A, B, .. are all functions, then these appear as, e.g. (= A B ..).

```
(GENERATE (CLAUSE INDEPENDENT INDICATIVE INTERROGATIVE
           NON-POLAR WH SUBJECT-FOCUS
           NON-MODAL TRANSITIVE PASSIVE ACTOR-UNSPECIFIED))

FR  1  (!PROCESS)
FR  9  (!SUBJECT !PROCESS)
FR 10  (!FINITE !SUBJECT !PROCESS)
FR 12  (!MOOD-FOCUS !FINITE !SUBJECT !PROCESS)
FR 16  (!FINITE !SUBJECT !PROCESS (= !QUESTION !MOOD-FOCUS))
FR 18  (!FINITE !PROCESS (= !QUESTION !MOOD-FOCUS !SUBJECT))
FR 24  (!GOAL !FINITE !PROCESS (= !QUESTION !MOOD-FOCUS
           !SUBJECT))
FR 26  (!TRANSITIVE !GOAL !FINITE !PROCESS (= !QUESTION
           !MOOD-FOCUS !SUBJECT))
FR 33  (!TRANSITIVE !FINITE !PROCESS (= !GOAL !QUESTION
           !MOOD-FOCUS !SUBJECT))
FR 34  (!PASSIVE !TRANSITIVE !FINITE !PROCESS (= !GOAL
           !QUESTION !MOOD-FOCUS !SUBJECT))

SQ  1
        ?
           !PASSIVE
           !TRANSITIVE
           !FINITE
           =>
              (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
              !PROCESS
SQ  2
        ?
           !TRANSITIVE
           ->
              ?
                 (= !FINITE !PASSIVE)
                 (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
              !PROCESS
```

```
ADDN  2
        ?
            !TRANSITIVE
            ->
               ?
                  (= !POST-SUBJECT !FINITE !PASSIVE)
                  (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               !PROCESS
SQ  1
        ?
            !TRANSITIVE
            ->
               (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               !PROCESS
ADDN  3
        ?
            !TRANSITIVE
            ->
               (= !POST-VERB !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               !PROCESS
COMP  2
        ?
            !TRANSITIVE
            ->
               (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               !PROCESS
ADDN  4
        ?
            !TRANSITIVE
            ->
               (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               (= !EN !PROCESS)
CONF  2
        ?
            ->
               (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               (= !TRANSITIVE !EN !PROCESS)
ADDN  3
        ?
            ->
               (= !POST-VERB !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               (= !TRANSITIVE !EN !PROCESS)
COMP  2
        ?
            ->
               (= !GOAL !QUESTION !MOOD-FOCUS !SUBJECT)
               (= !POST-SUBJECT !FINITE !PASSIVE)
               (= !TRANSITIVE !EN !PROCESS)
```

```
(!GOAL !QUESTION !MOOD-FOCUS !SUBJECT)

FNR 13  ((OR NOMINAL DEPENDENT))
FNR 14  (QUESTIONING (OR NOMINAL DEPENDENT))
SY 14  (PHRASE QUESTIONING (OR NOMINAL DEPENDENT))
SY  0  (PHRASE QUESTIONING NOMINAL)

(!POST-SUBJECT !FINITE !PASSIVE)

FNR  3  (FINITE-VERB)
FNR  6  (BE FINITE-VERB)
SY 21  (WORD VERB GRAMMATICAL-VERB NON-MODAL-VERB BE
          FINITE-VERB)

(!TRANSITIVE !EN !PROCESS)

FNR  8  (TRANSITIVE-VERB)
ENR  2  (EN-FORM TRANSITIVE-VERB)
FNR  7  (LEXICAL-VERB EN-FORM TRANSITIVE-VERB)
SY 22  (WORD VERB LEXICAL-VERB EN-FORM TRANSITIVE-VERB)
SY 17  (NON-FINITE-VERB WORD VERB LEXICAL-VERB EN-FORM
          TRANSITIVE-VERB)
```

Thus, the structure generated is



The program may then generate the features of the immediate constituents, using the function-realisation rules and systems, and then repeat the above process to determine the structure of the immediate constituents. The first stage of this is indicated above.

Of course, this is a particularly simple sentence and structure designed to make it easy to see what the program does, and it should be clear that considerably more complicated grammars can

also be handled. The generative process itself will not usually involve such straightforward intermediate structures or proceed so immediately to the final structure. For example, the generation of the structure of a sentence such as "Must it grow darker?", requiring five 'loops' of the structure-building rules before a structure compatible with all rules is obtained, is as follows:

```
(GENERATE (CLAUSE INDEPENDENT INDICATIVE INTERROGATIVE POLAR
          MODAL INTRANSITIVE ATTRIBUTIVE))

FR  1  (!PROCESS)
FR  9  (!SUBJECT !PROCESS)
FR 10  (!FINITE !SUBJECT !PROCESS)
FR 12  (!MOOD-FOCUS !FINITE !SUBJECT !PROCESS)
FR 20  (!MODAL !MOOD-FOCUS !FINITE !SUBJECT !PROCESS)
FR 27  (!MODAL !MOOD-FOCUS !FINITE !PROCESS (= !ATTRIBUANT
           !SUBJECT))
FR 28  (!ATTRIBUTE !MODAL !MOOD-FOCUS !FINITE !PROCESS
           (= !ATTRIBUANT !SUBJECT))
FR 29  (!COPULAR !ATTRIBUTE !MODAL !MOOD-FOCUS !FINITE
           !PROCESS (= !ATTRIBUANT !SUBJECT))

SQ  1
        ?
            !COPULAR
            !ATTRIBUTE
            !MODAL
            !FINITE
            =>
                !MOOD-FOCUS
                ->
                    (= !ATTRIBUANT !SUBJECT)
                    !PROCESS
SQ  2
        ?
            !COPULAR
            !ATTRIBUTE
            ->
                ?
                    (= !FINITE !MODAL)
                    =>
                        !MOOD-FOCUS
                        (= !ATTRIBUANT !SUBJECT)
                    !PROCESS
```

```
ADDN  1
        ?
            !PRE-SUBJECT
            !COPULAR
            !ATTRIBUTE
            ->
                ?
                    (= !FINITE !MODAL)
                    =>
                        !MOOD-FOCUS
                        (= !ATTRIBUANT !SUBJECT)
                    !PROCESS
SQ  1
        ?
            !COPULAR
            !ATTRIBUTE
            ->
                ?
                    (= !FINITE !MODAL)
                    ->
                        (=> !MOOD-FOCUS !PRE-SUBJECT)
                        (= !ATTRIBUANT !SUBJECT)
                    !PROCESS
SQ  2
        ?
            !COPULAR
            !ATTRIBUTE
            ->
                =>
                    !MOOD-FOCUS
                    (= !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
ADDN  2
        ?
            !COPULAR
            !ATTRIBUTE
            ->
                =>
                    !MOOD-FOCUS
                    (= !POST-SUBJECT !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
COMP  1
        ?
            !COPULAR
            !ATTRIBUTE
            ->
                =>
                    !MOOD-FOCUS
                    (= !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
```

```
ADDN  3
        ?
            !COPULAR
            ->
                =>
                    !MOOD-FOCUS
                    (= !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
            (= !POST-VERB !ATTRIBUTE)
SQ  1
        ?
            !COPULAR
            ->
                =>
                    !MOOD-FOCUS
                    (= !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
                (= !POST-VERB !ATTRIBUTE)
CONF  1
        ?
            !COPULAR
            ->
                (= !MOOD-FOCUS !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                !PROCESS
                (= !POST-VERB !ATTRIBUTE)
CONF  2
        ?

                (= !MOOD-FOCUS !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                (= !COPULAR !PROCESS)
                (= !POST-VERB !ATTRIBUTE)
ADDN  2
        ?
            ->
                (= !POST-SUBJECT !MOOD-FOCUS !FINITE !MODAL
                    !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                (= !COPULAR !PROCESS)
                (= !POST-VERB !ATTRIBUTE)
COMP  1
        ?
            ->
                (= !MOOD-FOCUS !FINITE !MODAL !PRE-SUBJECT)
                (= !ATTRIBUANT !SUBJECT)
                (= !COPULAR !PROCESS)
                (= !POST-VERB !ATTRIBUTE)
```

4. Conclusion

The obvious conclusion - that the mechanics of systemic grammar (as described by Hudson) are sufficiently well-defined to form the basis of a computer model - is, for linguistic descriptions, a significant one. However, the program also demonstrates that some rule-descriptions require clarification. For example, feature-realisation and function-realisation rules are implicitly unordered (since features and functions are unordered), or, more precisely, the rules are to be considered to apply simultaneously. This causes problems with those rules which prevent features being introduced (e.g. rules such as feature-realisation rule 32, which means "if !SUBJECT is present the realisation is as stated, otherwise the first function, i.e. !ACTOR, must not be introduced by any other feature-realisation rule"). The solution seems to be to reapply the rules, recursively, until a structure is produced which is compatible with all the rules.

More seriously, the expectation that the structure obtained is independent of the order of application of structure-building rules is not realised, at least for the grammar specified. For example, considering the second of the above generations and applying rules SQ 1, SQ 2, ADDN 2, SO 1, ADDN 1 (in that order) to the set of functions obtained by the feature-realisation rules, we generate

```
?
    !PRE-SUBJECT
    !COPULAR
    !ATTRIBUTE
    ->
        =>
            !MOOD-FOCUS
            (= !SUBJECT !ATTRIBUANT)
        (= !FINITE !MODAL !POST-SUBJECT)
        !PROCESS
```

This structure cannot satisfy both sequence rules, i.e. the generative process is blocked. Clearly, either the grammar requires modification or it does matter which order the structure-building rules are applied. Hudson (personal communication) has concluded that there are linguistic grounds for ordering structure-building rules, so that 'abnormal' cases precede 'normal' ones, with the latter only applying if the former had not already been applied. Whether it is possible to do so consistently requires further experimentation.

Further possible extensions to the work could involve trying to specify a lexicon so that the generative process ends up with a structure with words as leaves, and one could also attempt to apply the rules in reverse, i.e. to start with a string of words and produce a structural description. Both problems are, of course, very difficult ones.

Acknowledgement

The author is very grateful to Dr. R.A. Hudson University College, London) for his comments on this work.

References

Chomsky, N. (1965). Aspects of the theory of syntax. Cambridge, Mass.: MIT Press.

Friedman, J. (1971). A computer model of transformational grammar. New York: American Elsevier.

Halliday, M.A.K. (1961). Categories of the theory of gramma . Word 17, 241-292.

Halliday, M.A.K. (1970). Language structure and language function. In J. Lyons (ed.), New horizons in linguistics, London: Pelican.

Hudson, R.A. (1971). English complex sentences. London: North-Holland.

McCarthy, J. (1965). Lisp 1.5 programmer's manual. Cambridge, Mass.: MIT Press.

Power, R. (1974). A computer model of conversation. Ph.D. thesis, University of Edinburgh.

Self, J.A. (1975). CBLP - a computer based learning programming system. J. of. Inst. of Computer Sciences, in press.

Winograd, T. (1972). Understanding natural language. New York: Academic Press.