Natural Language Generation as a Computational Problem: an introduction[1]

David D. McDonald[2]
University of Massachusetts at Amherst

COINS Technical Report 81-33

(December 1981)

1. This paper, along with papers by Allen, Berwick, Kaplan, Sidner, and Webber, will appear in *Computational Theories of Discourse* edited by Brady, MIT Press, 1982.
2. The research reported here was performed while the author was a graduate student in the MIT Artificial Intelligence Laboratory. The Laboratory's artificial intelligence research is supported by the Defense Advance Research Projects Agency under Office of Navel Research contract N0014-75-C-0643. The preparation of this report was supported by the National Science Foundation under grant IST-8104984.

Research into the process of goal-directed natural language generation by computers is in its infancy. Until recently there has been no pragmatic pressure to go beyond the simplest ad-hoc generation facilities because the communications needs of the programs that would use the facilities has not required it. Theoretical accounts of generation have lagged accordingly since sophisticated theories of language use cannot be developed apart from equally sophisticated models of language users. Now however, the advent of expert programs in medicine, command-and-control, computer-aided-instruction, and similar language-intensive fields has made deep theories of language generation a necessity if these programs are to have an adequate ability to explain their conclusions and reasoning in a continually changing task environment.

The meager amound of computational research on language generation to date requires us to begin with simple questions with the goal of developing a general organizing theory. Without answers to the most basic questions about the process, analyses of specific phenomena of the sort that the other papers of this volume address in the context of language understanding cannot yet be profitably addressed in generation; rather we must first determine: What does language generation start with? What kinds of decisions are made and how are they controlled? What sorts of intermediate representations are needed? Modern theories of linguistic competence, though cast in a "generative" framework, are not suited to the job of goal-directed generation because their formal structure does not permit them to address the central problem, i.e. *how specific utterances arise from specific communicative goals in a specific discourse context.*

This paper is extracted from a much larger work, [McDonald 1980], which elaborates and argues for my computational theory of natural language generation. Since there is relatively little experience with natural language generation in the literature, we will begin with an exposition of some of the results that have been achieved using this theory with several artificial speakers, elaborating the linguistic and rhetorical problems that have been dealt with. This discussion will give the reader an idea of the kinds of problems that this research has concentrated on and where the focus of the theory lies. Following that we will consider what it means to have a computational model of generation and sketch the limitations that have been imposed on it. The main points of the model will then be presented, including a walk-through of one of the example outputs. Finally the model will be contrasted briefly with earlier generation techniques and the utility of the present computer program discussed.

1. Results for Test Speakers

It is a truism in artificial intelligence research that one cannot study thinking except by studying thinking *about* something in particular. This is true in the study of language generation: there is no such thing as generation in the abstract; one must study the generation of specific, well-developed artificial speakers performing in specific discourse contexts. The most fluent goal-directed generators of the past have all been based on a conceptually well-developed "speaker" program, e.g. the tic-tac-toe model of [Davey 1974] or the psychoanalytic patient of [Clippinger 1978], since it is only when the conceptual basis for the decision-making is well-grounded that stylistic linguistic decisions can be made on a sound basis.

The present research takes this one step further by generalizing the "linguistic component" of the generator to deal with more than one conceptual input representation. The question of "what do you start from" in generation has always been a vexing one: When studying language understanding, the input represention to the process (i.e. English text) is agreed upon by everyone and its details can be specified to whatever degree one likes. The psycholinguistically correct source for language generation on the other hand is utterly unknown and likely to remain so for some time; futhermore any variation in the details of the input representation will have considerable repercussions within the generation process (cf. [McDonald 1980] chapter four). Faced with this situation, I have (1) deliberately separated the conceptual and the linguistic phases of decision-making in generation into two modules connected by an explicit interface, thereby making the dependencies between them clear; and (2) tested the linguistic module with six different conceptual modules ("speakers"), four completed, two in progress, employing five different styles of conceptual representation.

Research on a separable linguistic component within the generation process is based on the hypothesis that linguistic decisions and representations within the process can be legitimately and profitably distinguished from conceptual ones. For this separation to be sensible, it must be the case (1) that the interactions between the "linguistic component" and the rest of the process can be specified precisely, and (2) that the extent of their shared assumptions about representations and contingencies is small (otherwise the linguistic component would have to be largely rewritten for each new speaker).

The relationship between the linguistic component and the larger system is sketched in figure 1.

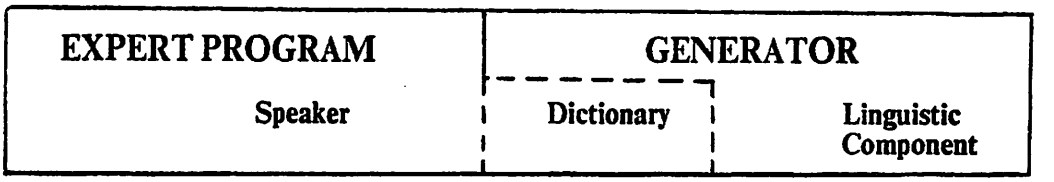


Figure 1 A Coarse View of the Total System

The *expert program* is what human users really think they are talking with; the generator is just part of a "natural language interface". The expert is grounded in a particular *conceptual domain* such as internal medicine or petroleum geology. It is expected to have no linguistic knowledge of its own; instead, any domain-dependent knowledge about how to answer questions, what information to include, what to leave out as obvious, knowledge about how to give explanations at the appropriate level of abstraction, or comparable discourse abilities is located in a *speaker component* which may or may not be a physically independent part of the expert program. The speaker assembles a *message* for input to the linguistic component, describing the goals it wishes to achieve with its utterance using whatever representation is convenient to it and the expert program. Messages are decoded by a *dictionary* compiled specially for each new representation; the dictionary is where the knowledge of what natural language phrases could be used to realize the components of the messages is stored.

The *generator* consists of the "speaker" (with its dictionary) and the "linguistic component": the speaker deciding roughly "what to say" and the linguistic component deciding "how to say it" and orchestrating the actual production. A distinction like this is common to nearly every generator that has been developed. (See for example, [Simmons & Slochum 1972; Goldman 1974 ; Davey 1974; Moore 1981], but compare [Shapiro 1975].) For existing expert systems, the notion of a separable "speaker component" is only a convenient fiction, the knowledge of the speaker having been incorporated *ad hoc* into the expert proper; nevertheless, if future experts are to have the fluency and versatility they will require, then this special kind of conceptual knowledge that I am attributing here to a "speaker" will have to be incorporated a theoretically sound way. In any event, it will be convenient to think in terms of the speaker as the part of the expert system that makes all the decisions about what to say, providing the input to the linguistic component.

1.1 The Different Input Representations

In this section we will go briefly through the six input representations that have been explored and then look at a sample of the results that have been achieved, giving examples of generated texts and discussing the linguistic phenomena involved in their construction.

The speakers/expert-programs have been by necessity artificial and minimal: The burden of this research was intended to be on the linguistic problems of generation rather than the conceptual ones, and the expert programs in existence at the time this research was begun were not sufficiently sophisticated to

motivate the linguistically interesting constructions of English and thus could not be used. (Relevant English constructions include: embedded clauses, ellipsis, pronominal and non-pronominal subsequent reference, arbitrarily embedded wh-movement, and thematic relations such as focus and given/new.) Consequently each speaker had to be built from scratch, and was elaborated only as far as was needed to motivate the English it was intended to illustrate. In the completed speakers, not much more than the examples shown was ever actually developed. Below is a summary of the speaker programs used according to the type of conceptual representation they employed; a detailed description of the first two will follow.

Predicate Calculus Well-formed formulas in the predicate calculus, in isolation and in natural deduction proofs, were supplied directly as the linguistic component's input, e.g. from $\forall(x)\ \mathbf{man}(x) \rightarrow \mathbf{mortal}(x)$ the component produced: *"All men are mortal"*. This domain presented an opportunity to study the decoding of message-level conventions such as expressing quantifiers as determiners or type predicates as class nouns, as well as discourse coherency and the symbolic analysis of possible realizations.

Assertions in PLANNER-style Data-bases A description of a semantic net was supplied to the generator as a set of simple relational assertions about the nets components parts. One net corresponded a multi-paragraph text, one paragraph per node, ordered according to a depth-first scan. This domain provided an opportunity to produce large texts without developing an elaborate expert program, and provided a study of stylistic variation, the use of the thematic relations focus and given/new, and of the use of ellipsis and indefinite anaphora including the automatic collapsing of conjoined predicates at the message-level.

OWL The language OWL, developed by William Martin [Hawkinson 1975], is a compositional representation specifically designed as the target output formalism of a natural language understanding system (and therefore able to represent naturally the kinds of underspecification, ambiguity, quantification, etc. found in natural languages). The work on this domain was done by an beginning MIT graduate student, Ken Church, part-time during the fall of 1978. The inputs to the program were literal procedures taken from DIG, the digitalis therapy advisor developed originally by Silverman [1975] and reimplemented for explanations by Swartout using OWL [1977]. The resulting texts from Church's work (which will not be shown) were comparable to, though not quite as smooth as, the texts originally obtained by Swartout. Church's work demonstrated what had been suspected earlier, namely that because its one-pass control structure is biased to expect rhetorically pre-planned input, this linguistic component is not a good place to stage large-scale reanalyses of a domain's conceptual structure.

FRL FRL, "Frame-oriented, Representation Language" was developed by Goldstein and Roberts [1977] as an experimental implementation of "frame" ideas of Minsky [1974]. It was used by Winston as the representation for his program for making and evaluating analogies [Winston 1980]. A dictionary was compiled for Winston's database on the play "Macbeth", from which texts were directly produced describing the actors and major scenes. Winston imposed a rigid "case-frame" discipline on the fields of his FRL frames, making them very easy to translate into English. This made it possible to concentrate instead on the

coherency of the text (as in the semantic net domain), and to develop a battery of general linguisic transformations to deal with propositional attitudes, subordinate clauses, sentence-level adjunction, and thematic focus, and to study explicitly planned cataphor and subordination (as in *"Because Lady Macbeth persuaded him to do it, Macbeth murdered Duncan."*).

KL-ONE KL-ONE is a highly structured semantic net formalism under development at BBN [Brachman 1978; Woods 1979]. The work on this representation is still very much underway and has been initially reported in [McDonald 1980]. In their primary generation application KL-ONE nets are used as the knowledge base of a tic-tac-toe program, modeled after the work of Anthony Davey [1974], that gives fluent commentaries of games of tic-tac-toe that it has either played or read. It provides an opportunity to experiment with discourse-level planning, and to study how rhetorical intentions can control descriptions (e.g. whether to say *"the corner opposite the one you just took"* or just *"a corner"*). A second project using the KL-ONE representation began in the fall of 1981 with the task of producing paragraph-length English descriptions of natural scenes starting from the output of an A.I. scene-understanding system. This domain focuses on the problem of planning vocabulary selection, particularly how the choice of spatial description may be constrained by grammatical context.

Next we will look at some example input and output from the first two test domains and consider the linguistic problems that had to be faced in order to produce them. After this section, we will discuss the principles behind the linguistic component and follow through an example in detail.

1.2 The LOGIC Domain

In any study of language generation, it is important that the message-level representation with which the process starts be credible. It would be questionable, for example, whether a program that started from a dictionary of fragments of English sentences could be said to have solved any significant problems. The predicate calculus, on the other hand, is a very credible message representation: it is an accepted, comfortable "internal representation" for the programs of a large part of the artificial intelligence community; it has a universally agreed upon interpretation; and it is sufficiently unlike natural language in form that demonstrations of the work that one's linguistic component has done are readily available.

The *logic domain* consists of a representation for well-formed formulas in predicate logic, routines for translating formulas typed by a user into this representation and storing them, and a dictionary with fixed entries for the logical connectives and inference rules and a set of conventions for new entries that the user may write for particular predicates, constants, and typed variables. There is no speaker or expert program *per se*, all of the interpretation of conventions and application of discourse heuristics that a "speaker" would do being embedded directly in the entries of the dictionary.

The original work with the logic domain consisted simply of presenting the program with a single well-formed formula ("wff") and having it produce an English rendering. For example

∀(block) ∀(surface) space-for(surface,block) ↔ (table(surface) ∨ cleartop(surface))

was rendered as:

"There is space on a surface for a block if and only if that surface is the table or it has a clear top."

Different conventional interpretations of formulas were experimented with, originally under explicit control of the designer and later under program control using both lookahead at the lingustic decisions and simple tests of the logical structure of the expressions to determine whether an interpretation would go through. The same formula, say: **"∀(x) man(x) → mortal(x)"**, can be understood conventionally and rendered as: *"All men are mortal"*, or understood literally and rendered as: *"For any thing, if that thing is a man, then it is mortal"*.

It does not take long, however, to exhaust the linguistic insights to be gained from looking at single formulas in isolation. A predicate calculus formula is underdetermined with respect to the more sophisticated forms of reference and quantification supported by natural languages, and its connectives and predicates can usually be given many equally plausible renderings. When formulas appear in isolation, there is no motivation for using one rendering or one interpretation of a quantifier over another.

One way to provide the needed motivation is to look at formulas in the context of a proof. Figure 2 shows a natural deduction proof followed by the text that the logic domain's dictionary selected for it. (The first line is a statement of the "barber paradox" created by Bertrand Russell as a popular rendering of the set of all sets paradox.)

line1: premise
$\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y)))$
line2: existential instantiation (1)
$barber(g) \wedge \forall y(shaves(g,y) \leftrightarrow \neg shaves(y,y))$
line3: tautology (2)
$\forall y\ shaves(g,y) \leftrightarrow \neg shaves(y,y)$
line4: universal instantiation (3)
$shaves(g,g) \leftrightarrow \neg shaves(g,g)$
line5: tautology (4)
$shaves(g,g) \wedge \neg shaves(g,g)$
line6: conditionalization (5,1)
$\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y))) \rightarrow (shaves(g,g) \wedge \neg shaves(g,g))$
line7: reductio-ad-absurdum (6)
$\neg\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y)))$

Assume that there is some barber who shaves everyone who doesn't shave himself (and no one else). Call him Giuseppe. Now, anyone who doesn't shave himself would be shaved by Giuseppe. This would include Giuseppe himself. That is, he would shave himself, if and only if he did not shave himself, which is a contradiction. This means that the assumption leads to a contradiction. Therefore, it is false, there is no such barber.

Figure 2 The Barber Proof

The lines of the proof are passed to the program in sequence; the English text selected for earlier lines provides a discourse context to narrow the choices available to later ones directly controlling subsequent references to constants, variables interpreted as generic references, and predicates and formulas used as descriptions. Further motivation for text choice is provided by the labels that are attached to certain lines to reflect their role in the structure of the proof (e.g. *"the assumption"* or *"a contradiction"*), and by the logical inference rules that derived the lines: a large part of the rendering of the proof must be an explanation, guided by the inference rules, of how each line follows from the earlier ones.

The proofs that were used in the logic domain were selected from a set of proofs that had been used by Daniel Chester [1976] in virtually the same task. The choice was made deliberately to permit a direct comparison of the output of the two systems on the same material—something that is rare in studies of language generation. Chester's version of the "barber proof" is as follows:[3]

3. My source for Chester's results is a personal communication with him in November of 1975; the major effort on the logic domain was completed in December of 1977.

Suppose that there is some barber such that for every person the barber shaves the person iff the person does not shave himself. Let A denote such a barber. Now he shaves himself iff he does not shave himself, therefore a contradiction follows. Therefore if there is is some barber such that for every person the barber shaves the person iff the person does not shave himself then a contradiction follows. Thus there is no barber such that for every person the barber shaves the person iff the person does not shave himself.

Chester's program belongs to the "direct translation" school of natural language generation systems (see [Mann et al. to appear]). It produced the paragraph above by recursively replacing the formulas of the proof with English text (after editing it for production), entirely on the basis of local properties of the formulas. The lack of contextual input to the program's realization decisions is reflected in its minimal treatment of subsequent reference and the occasional abruptness of transition from line to line.

At this point, I will use the example of the barber proof to point out some of the accomplishments that are embodied in the current version of my generation program.

The ability to go beyond the literal content The program processes a proof by realizing its formulas and subformulas one at a time in top down order (i.e. the construction axioms of the predicate calculus are followed). The formulas are not translated mechanically, but rather at each step along the way, a context-sensitive decision is made as to how (or whether) the major logical connective (or inference rule) is to be realized. and which (if any) of the subelements of the formula are to be involved in that realization. Line three of the proof, for example, has no corresponding sentence in the text because we can assume that such a step in the proof would be made automatically by the audience. (This is an implicit, conventional assumption: there is no simulation model of the user.) Line four, on the other hand, has been expanded into three sentences because the logical substitution of a second instance of the same constant is assumed to be liable to confuse the audience. The three sentence subargument is constructed by putting a special rhetorical twist on the formula of line three (to define the set), adding a new formula based on the variable being substituted (sentence four), and concluding with the formula from line four.

The logical conjunction in line one is interpreted as a conventional way of defining the type of the variable "**x**". Similarly the two quantifiers in that line are realized in the determiners of their variables (*"some barber"*, *"everyone"*) rather than as *"for"* phrases.

Subsequent reference Knowing when *not* to use a pronoun is very important in the production of understandable texts. Thus while the barber is identified and given a name in first two sentences, he is not pronominalized in the third and fourth because those sentences are part of a new new discourse structure (the "subargument" composed to ease the transition to line 4) where the discourse focus is on the universally quantified variable "**y**" rather than on the barber *"Giuseppe"*. When the focus shifts to him in sentence five as a result of the use of the intensifying reflexive (*"Giuseppe himself"*), he can then be pronominalized in the four instances in sentence six. (N.b. the name *"Giuseppe"* was picked arbitrarily.)

Descriptions may be "pronominalized" as well as references. At the end of sentence one, the original description of "y" (i.e. *"everyone who doesn't shave himself"*) is recapitulated in the description of the complement set as: *"no one else"*. Then in the final sentence, the original complex description of the barber is reduced to just the adjective *"such"*.

Functional labels The premise functions in the proof as *"an assumption"* that is to be shown to be false because it leads to a contradiction. Since this role is known to the audience (we began by saying *"Assume that... "*), we can use the label later (sentence seven) as a succinct reference to the entire first line. The logical schema "A ∧ ¬A" is similarly labeled as *"a contradiction"*. Part of the concept of a label is the ability to include a literal rendering of the labeled expression as an appositive (final sentence). In the logic domain's dictionary, appositives are triggered if the last literal rendering was not in the same paragraph or, as in this case, if the line is the conclusion of an argument.

Context sensitive realizations Part of the linguistic context that is produced to guide later decisions is a rhetorical description of the discourse structure. The different terms of this structure will guide decisions at syntactic and morphological levels: in sentences one and three a contraction is used (*"doesn't shave"*) while the same logical structure in the *formal* context created by the *conclusion* sentence of the subargument is not contracted (*formal* being an experimental rhetorical feature in the grammar). Similarly the connective ↔ is spelled out in a formal context (sentence five), but in an unmarked, informal context, it is understood as a restriction on a variable and expressed as a relative clause. In another case, the same quantified variable ("y") is realized in the unmarked context of sentence one as *"everyone"*, but when marked in sentence three as identifying a set it is realized as *"anyone"*.

Part of the discourse context is the distance between phrases. When a contradiction is deduced from the immediately previous line, as in line five, the identification of that deduction is given in the most direct way possible by adjoining a relative clause to the last sentence; when the dependency line is much earlier (as in line six), the formula from the line is repeated and the phrase *"leads to"* is used.

Attempts to avoid ambiguity In sentence one, the interpretation of "↔" as a restriction on a variable's range must include some phrase to indicate that the entire range has been specified and not just a part of it. Consequently the "iff-entry" is designed to say "<restriction> *and* <complement of restriction>". (An equivalent technique would have been to replace the word *"everyone"* with *"all and only those men who... "*.) Because the presentation of this combined restriction should be done carefully, a special monitoring routine is activated in an attempt to avoid introducing scope ambiguities in the conjunction. On the basis of the point where the conjunction is attached (i.e. as the direct object of *"some barber who shaves ___"*), the projected contents of the second arm of the conjunction (a noun phrase), and the fact that the first arm has ended with a direct object, the monitor decides that it is possible that the second arm will be missinterpreted as conjoining with the more immediate, lower direct object rather than with the intended one. It causes the parentheses to be added around the second arm as one way available to it in this case to try and forestall missinterpretation.

1.3 PLANNER-style Assertions

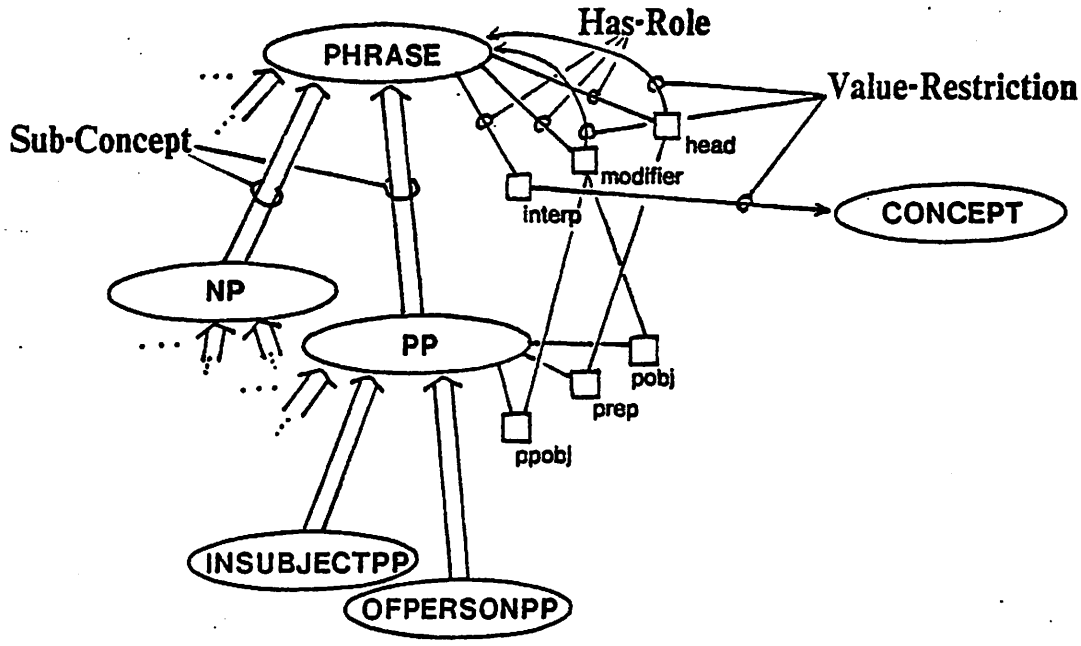
The source data structure in this domain was a KL-ONE network reimplemented as a set of binary relations expressed as PLANNER-style assertions. A KL-ONE net consists of a set of named objects of various types (only "concept" and "role" are shown here), linked together by the relations: "subconcept", "has-role", "value-restriction", and others not shown. This domain took the network as input and produced an English description of its literal contents; that is, rather than interpret the net as a representation of certain facts (e.g. *"every phrase has a head, a modifier, and an interpretation... "*), it is interpreted at its literal level as a collection of KL-ONE objects (e.g. *"The concept phrase is the top of the net, it has a head role that..."*).

Figure 2 shows the first paragraphs of the text constructed for one of the development networks in use at Bolt Beranek and Newman Inc. ("BBN") during the spring of 1979. (It represents a first pass at a conceptualization of an English grammar.) The text was created by scanning the net depth-first following its "subconcept" links, devoting one paragraph to each concept. Each paragraph mentions (or assumes—see below) three facts about about its concept: (1) the name of the concept(s) it is a subconcept of, (2) the names of its "roles" and the "value-restrictions" they are subject to, and (3) the names of its own subconcepts if any. (The fact that each paragraph will present a new concept is taken to be already known to the audience, and as a consequence, the information that, e.g., *"phrase is a concept"* is omitted as already given.)

Varying the paragraph structure The few paragraphs shown in the figure are sufficient to illustrate the stylistic heuristics that the dictionary for this domain incorporates. (Like the logic domain, this domain had no speaker as such; its messages were comprised directly of KL-ONE nets or coherent subnetworks.) In each of the first three paragraphs, the presentation of the concept's roles and their value-restrictions is given in a different style. It is done by varying the rhetorical pattern of the description according to the number of roles the concept has. In the first paragraph, "phrase" has three roles and the style chosen puts each role in a separate sentence: "<role> *must be* <value-restriction>". The second paragraph's concept has more than three roles, leading to the use of a summarizing sentence to identify them as its roles before giving their value-restrictions. The third paragraph, with only two roles, uses sentences based on the "has-role" relation, with each value-restriction embedded as a relative clause.

Omitting "given" information Note that the second, third, and fourth paragraphs do not start with a sentence about what their concept is a subconcept of. This is because that information appears in the text already (in the last sentence of each previous paragraph) and the dictionary entry that would make the decision to include that information decides that it will be still remembered and thus would be redundant if included. Similarly in the second paragraph where there is a summary sentence listing roles of the concept pp's, the "has-role" facts have been left off of the later sentences since to leave them in would have led to an unacceptably redundant text.

Varying descriptions with context The noun phrases constructed to describe roles vary along the same lines as paragraphs, i.e. they include facts or leave them out depending on what facts have already appeared in



Phrase is the top of the net. Its interp role must be a concept, and its modifier role and its head role must be phrases. Its subconcepts are pp, np, adjunct, indobjclause, and word.

Pp has the roles: pobj, prep, interp, and ppobj. Pobj must be a np, prep a prep, interp a relation, and ppobj a pp. Pp's subconcepts are ofpersonpp, insubjectpp, locationpp, and aboutsubjectpp.

Ofpersonpp has a pobj role which must be a humanp, and a prep role which must be an of.

Insubjectpp's pobj role must be a subjectnp, its preprole an in, and its interp role a subject.

...[[further paragraphs for the rest of pp's subconcepts]]

Np is another subconcept of phrase...

...[[further paragraphs for the rest of phrase's subconcepts and the subconcepts of each of those in turn]]

Figure 3 Describing a Semantic Net

---

their paragraph and what remain to be given. Thus we go from using just a name to introduce a role (paragraph three) to giving the concept that owns it, its name, and the fact that it is a "*role*" (in paragraph four).

Using ellipsis Throughout the example text, grammatically-driven ellipsis is applied to reduce redundant verbs (paragraph two), and to merge relations with common arguments (paragraph one). These are general purpose transformations, triggered by the syntactic and lexical properties of the texts, independently of the content of the relations involved.

2. A Computational Model

The ability to speak is as natural to us as the ability to see or to use our hands to grasp objects. We are fast, we are accurate, and we are unaware of the mechanics of how we do it.[4] As easy as it is for us to speak, we know from linguistic and ethnomethodological analysis that the process is complex. Even if we leave aside the question of how we arrive at the thoughts behind our words and look just at the "linguistic" part of the process—selecting words and constructions, applying grammatical rules, and producing the words in sequence—it is clear that very sophisticated rules are being followed. Somehow we select one lexical/syntactic combination from the many possible alternatives, managing to attend simultaneously to the potentials of the different constructions, our multiple goals, and the constraints arbitrarily imposed by our grammar. We follow conventions of direct utility only to our audiences and actively maintain elaborate coherency relations across large stretches of discourse.

Our ability to do all this with such facility needs to be explained. For this, a static description of the rules being followed will not be sufficient: we must explain what it is about the way these rules are represented and manipulated that insures that the process of language production is tractable and gives the process the character that it has. In short, we must develop a computational model: a simulacrum whose processing steps and representational devices when viewed from the intended level of abstraction we take to be isomorphic with those operating in the human mind.

It is important to appreciate that by "computational model", we do not simply mean a program whose input/output behavior matches that of people (though that in itself would be a considerable accomplishment). The internal structure of the program—the reasons *why* its input/output behavior is what it is—is critical to its value as a model. This is comparable to the requirement for "strong" rather than "weak" equivalence in constructing a grammar for a natural language. If the model is to be truely sucessful (and ultimately to be a source of testable predictions), its behavior must follow inescapably from its structure rather

4. The normal speaking rate for English is approximately four syllables per second or 160 words per minute. (The Guiness Book of Records speed record for reading English is 400 words per minute.) A study by Labov [1966] has shown that 75% of everyday speech is grammatical by any criterion. If general rules for ellipsis and self-editing are added, this figure rises to 90% for non-academic speakers talking about everyday experience. Introspective reports of production appear to go no deeper than mentally "hearing" full phrases (or alternative words) at a time. We appear to have no conscious access to any of the actual assembly processes such as the sequencing of the words or their morphological specialization.

than from stipulated rules; it will stand as an explanation of the behavior because any device with comparable structure would be incapable of behaving otherwise. If one can then independently show that the human language faculty is structured in the same way as the model (perhaps by comparing the kinds of errors that the two systems make), then one will have explained why people have the modeled behavior. Consequently, if our model is to be compelling, we must limit its computational power very carefully. A computational model that permitted the use of arbitrary procedures (e.g. a Turing machine) would not be interesting as the basis of a theory because all that it would explain would be that language production was computable: something we already believe. We must look instead for the weakest model that can do the work: a model from whose computational properties the characteristics of human language production would inexorably follow. By doing this, by restricting the kinds of behavior that our model is capable of, we can extract non-trivial predictions from it and make it subject to empirical tests. (Neither of which I intend to do in this paper. As will be clear, the model already meets a number of "obvious" psychological criteria such as sequential production and indelibility (see also [McDonald 1980]). Its first non-obvious application is expected to be in a theory of the mechanisms behind certain naturally occuring "speech errors", particularly exchange errors and blends. (See [Garrett 1980] for an extensive description of speech-errors.) This work, however, is still in progress.)

2.1 Characterizing the Problem

What computational problem is the mind solving when we talk? "Talking" is of course a loose term used to cover many kinds of activities, each likely to have its own requirements in processing time, necessary memory, possibilities for editing, or conscious involvement. We know intuitively that there is an enormous difference in behavior between, say, writing a careful essay and holding a fuzzy conversation over breakfast; so much so that there is little reason to believe *a priori* that they pose identical problems to the human speaker. In the present research, I have focused on *immediate speech*, spoken (or written) without rehearsal and with only a rough conscious knowledge of what will be said next. There is introspective evidence to suggest that this mode of speech is primary since even in deliberate writing where there is ample opportunity for editing and planning, it is the common experience that phrases and even multible sentences "spring to mind" as immediate percepts without any conscious effort having been made to form them from their constituent parts.

Given this restriction on the mode of speech to be considered, I take the core of the "problem" for the mind to be the re-expression of a delimited, deliberately selected "packet" of information (including references, propositions, descriptions, and probably specific rhetorical instructions) from its original form in the mind's internal representation into a constrained, fixed-format language (e.g. English) according to a fixed, context-free, conventional mapping.

2.2 Language Generation as Decision-making

What are to be the primitive operations of the model—at what "grain size" will it characterize the generation process? Following the lead of systemic grammarians such as Halliday [1966,1970] and Winograd [1973], I view the output of the process—the natural language text—as the result of a series of decisions, the set of possible consistent decisions being determined by the language's grammar. The most relevant aspects of a generator will then be how it goes about making those decisions, which is what the theory is to determine. In particular: (1) what kinds of decisions are to be made, what prompts them and what is the nature of their output; (2) what kinds of information the decisions require and how that information will vary in its accessibility and form according to the state of the process; (3) what dependencies there are between decisions and how they influence the overall control structure (e.g. are decisions made nondeterministically or are they necessarily ordered?); (4) to what extent the results of previous decisions and the foreknowledge of planned decisions are a part of the generator's state, i.e. is this kind of information explicitly represented and accessible to current decision-makers?

2.3 Restrictions on the Model

Given the problem of translating a packet of expressions/instructions into a highly constrained, fixed format language giv .i a context-free mapping (i.e. the translation dictionary), there are many known ways we could use to solve it: Approaches have ranged from nondeterministic optimizers that worked on whole paragraph-sized texts at once [Moore 1981], to programs that have attempted to model stream-of-consciousness and were liable to interupt themselves with new plans or constraints at every phrase [Clippinger 1978]. All approaches have in common the notion that the input packet or "message" will be decomposed into its component elements; that the elements will be looked up in the dictionary and a context-sensitive decision made as to how they can be realized in the target language; and that these realizations, subject to the constraints of the grammar and the overall goals of the message, are pieced together into the utterance. The approaches differ in nearly every other aspect, e.g. how large an utterance to construct at once, how to control the process and order the decisions, or how to represent the grammar and implement its constraints.

In the interests of narrowing the field of candidates, and because I believe that the resulting model is both more perspicuous to the engineer and more interesting to the psychologist, the following additional computational limitations have been stipulated in my theory.

On-line Operation The input message is viewed by the linguistic component as a stream of elements, the specific order and chunking of the "message elements" being dictated by the dictionary entries designed for that particular domain. The component may be conceptually (though not literally) decomposed into *two transducers* cascaded together, the first taking the next element of the message stream and converting it into a surface structure phrase attached to the tree at the point where the message element was, and the second then walking that phrase and producing the text from it (see section 4.1 below). The two transducers are constrained to operate "on-line", that is, the output from the first transducer must be completely consumed

by the second before the first transducer moves on the next message element at the same level.

Indelibility The decisions of the first, "realizing" transducer are exaustively represented in the surface structure phrases that it produces. The actions of the second, "tree-walking" transducer are then completely dictated by the structure and annotation of those phrases. *Surface structure is indelible*, i.e. once a phrase has been constructed and incorporated into the ongoing surface structure tree, it can not be removed or edited (though it may be augmented). As a consequence of indelibility and the fact that the surface structure is organized as a strict tree without loops, the process will not backup—it is impossible to retraverse earlier sections of the tree once the realizing transducer has past through them. (This same stipulation has been applied to the recognition of phrases by a parser [Marcus 1980] with intriguing results.)

Locality Decisions may only make reference to contextual information that is local to them at the position within the tree where they occur. There is no mechanism available in the model that would allow a decision to scan the tree for information; all potentially relevant information must be expressly recognized as such and specific provisions made in the grammar to make it available to decision-makers via locally defined and updated variables. The effect of this stipulation is to restrict the information available for a decision to no more than would be available to a parser using an LL(k) grammar.

Real-time The overall process must perform its computations in *quasi-real time*; that is, the number of operations that take place between the consumption of any one message element in the stream and the next or between the output of two sucessive words must be no greater than some fixed maximum unrelated to the size of the input or output streams. This is a stronger time bound that the usual one of linear time, and reflects the intuition that the process always proceeds at a constant rate.

3. The Relationship Between the Speaker and the Linguistics Component

3.1 'Messages'

If the linguistic component was used with only one speaker/expert program, then there would be no need for an elaborate interface between the two: all of the speaker's conventions and the linguistic component's conventions could be integrated and the responsibility for obeying them distributed evenly in a seamless merger of the two decision-makers. Explicit messages (and thus conventions for representing messages) would be needed only when it was necessary to represent goals that refered to the message itself (such as "don't be long-winded" or "don't use technical vocabulary"). But of course the opposite is true: there are many qualitatively different expert programs to be linked with a common generator; consequently, the linguistic component is a distinct module computationally as well as conceptually, and a uniform interface is required to smooth over the differences between domains.

Functionally, the linguistic component lies on the path between the speaker and its audience. The speaker decides what it wants to say, constructs a representation of its goals and references—a *message*—and passes it to the linguistic component. This transfer of an explicit expression is required simply because the linguistic component is not telepathic: it does not automatically know what a speaker will want to say, nor, since it works with many speakers, can it even have *a priori* assumptions about the kind of things that are going to be said or how they will be represented internally in the speaker/expert program. These must instead be spelled out (1) in the *dictionary*: which holds the information on how to interpret the expert's representation element by element to determine its linguistic correspondences and relevant substructure, and (2) in a set of *interface functions*: which know (a) how to link up an individual element of a message to the appropriate dictionary entry, and (b) how to answer certain idiosyncratic linguistic questions for the message elements such as their "person and number" or whether they act as references or descriptions.

Both the dictionary and the interface functions must be specifically designed for each new speaker/expert program; they are the repository of all the information required to adapt the linguistic component to such new domains. Given this modularity, expecially the functional interface, we can be flexible about the choice of formal representation for a message; we can use whatever representation is convenient for the speaker's planning, typically the representation used in the expert program (e.g. as described in section 1.1).

There is no presumption that messages should result in texts of any fixed size. With the present test speakers, single messages have produced texts ranging from single exclamations to multi-paragaph discourses. Neither do messages have to be equated with turns in a conversation since the linguistic state of the component is preserved between activations and a text can be "picked up where it left off".

Structurally, messages have fallen into two broad classes. The simplest just consist of pre-existing expressions taken directly from the expert's data base. (All of the completed test speakers fell into this class.) The data base expressions become "instructions for what to say" through the interpretation provided by their dictionary entries, and the result is a fairly literal rendering of the expression, the structure of the output text following the compositional structure of the input expression. Below is an example expression from Winston's data base of Shakespearian plays (given in FRL), followed by the text that the linguistic program generates for it using the dictionary for that domain.

```
(ma (ako (story))
    (part (macbeth)
          (lady-macbeth)
          (duncan)
          (macduff))
    (subpart (heath-scene)
             (murder-scene)
             (battle-scene)))
```

"'Macbeth' is a story. It has four characters: Macbeth, Lady Macbeth, Duncan, and MacDuff, and three scenes: the heath scene, the murder scene, and the battle scene."

Figure 4 A Simple Message and Its Output

In a simple "direct translation" message such as this, the linguistic component has some ability to simplify and smooth the text by following default rules keyed by the local linguistic structure as the text is built; however, all of the choices of ordering and level of detail are fixed by the internal structure of the data base expression. Since the structure of the data base is determined by whatever is convenient for the expert's internal computations rather than by considerations of text planning, messages of this sort are necessarily limited in the applicability. (See [Swartout 1981] for an extensive discussion of this problem.)

The second class of messages are those that are deliberately planned by the speaker and involve *relations* among the expert's expressions that are specific to that speech event and may involve special rhetorical relations (instructions to the linguistic component) that have no counterpart elsewhere in the expert. The formal structure of these messages will again be whatever is convenient computationally for the designer of the speaker component, typically an extension of the representation already in use in the expert, with the interface functions adapted to match. Figure 5 shows a handcrafted example of a planned message in Winston's domain. The development of planning programs that can take advantage of the abstract planning vocabulary that a linguistic component such as this one can support (e.g. instructions like "focus", "sequence", or "constrast"; note for example that the sentence structure of the output text was determined dynamically by the linguistic component rather than given in the message) is the subject of on-going research by the author and others especially [Cohen 1978; McKeown 1980]. Experimental messages such as the one in figure 5 are aimed at determining what degree of rhetorical abstraction is plausible and how much foreknowledge of the linguistic properties of expert's expressions is required in planning at this level. That a planner can in fact be designed at this level of modularity that will produce indelibly realizable messages is a hypothesis that underides most of the work on this lingustics component.

```
(message1 (sequence (macbeth (murder (duncan))) ;"murder-ma"
                    (macbeth (become (king))) ;"ma-become-king"
                    (lady-macbeth (persuade (macbeth (action murder-ma)))) ;"persuade-ma"
                    (lady-macbeth (hq (ambitious)))) ;"ambitious-lm"
          (time-frame (before-time-of-speech))
          (focus (macbeth))
          (ancilary-facts ((murder-ma (motive (ma-become-king)))
                           (persuade-ma (purpose (cause (murder-ma))))))))
```

"Macbeth murdered Duncan in order to become king. He was persuaded to do it by Lady Macbeth who was ambitious."

Figure 5 A Planned Message

In summary, what matters about a message is not the notation that is used but what it specifies and what it leaves to default. The structure of the message (as interpreted by the dictionary) directly determines the order in which the linguistics component will break it down and realize its elements; in simple speaking situations, literal expressions from the expert's data base may be the best messages—implicitly wrapped in the directive "describe these objects and relations"; however, as the situations become more complex and less predicatable, a full-scale, rhetorically knowledgeable planner will be needed to compose messages as the speaker's goals and discourse context demand.

3.2 Run-time Relationships

From the point of view of the speaker/expert program, the linguistics component is a *subroutine*, a subprocess that the speaker explicitly activates to realize an individual message. It is only activatable by the speaker, i.e. it has no independent existance as a parallel process (though the history of the discourse is contiguous across activations), and once activated runs to completion independently of the rest of the system. Its internal state is a black-box, not designed to be monitored, interupted, or edited (though it could be "shut-off" and completely restarted).

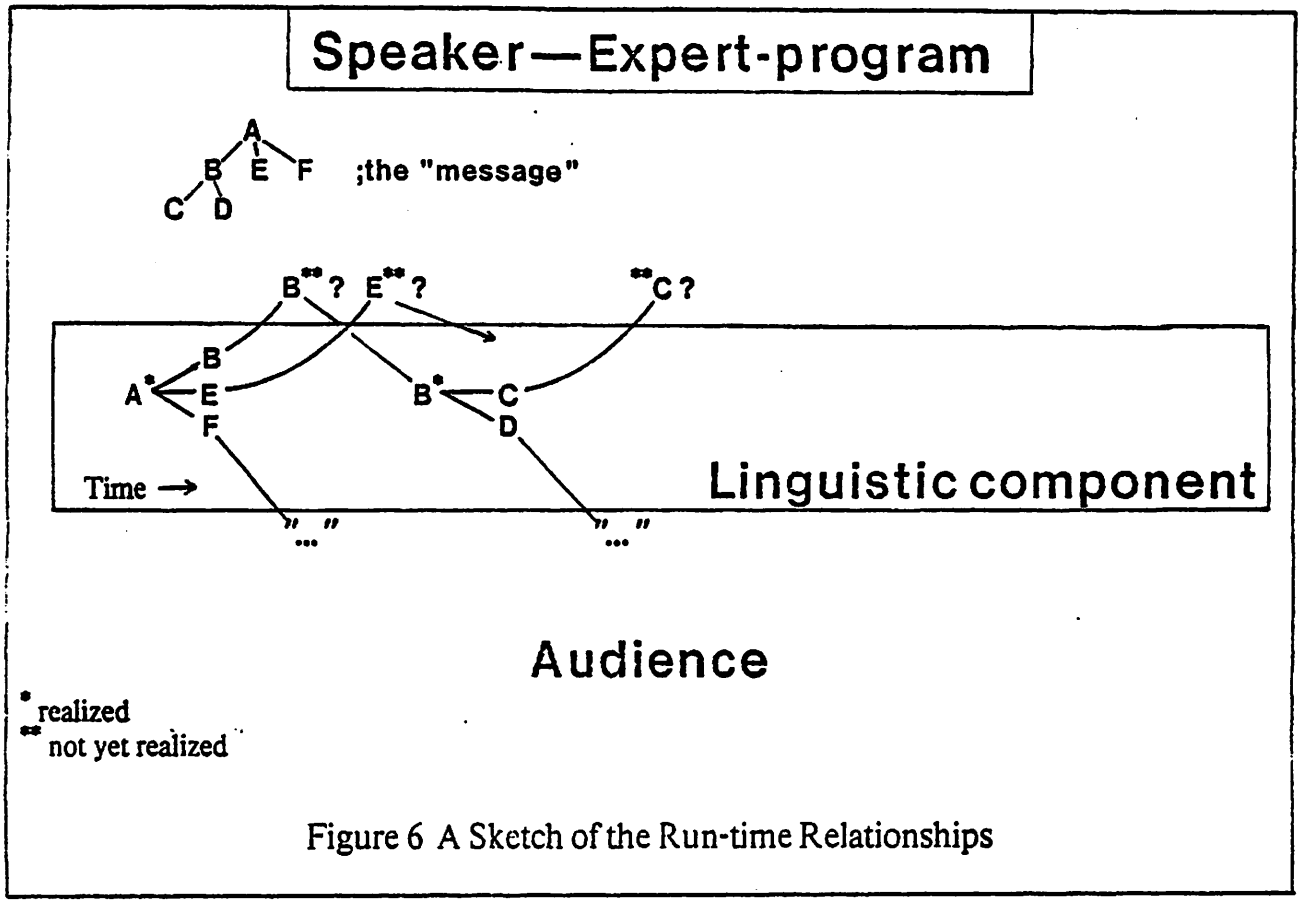


Figure 6 A Sketch of the Run-time Relationships

Though independently controlled, the linguistic component is not cut off from the speaker while it is processing. Figure 6 shows diagramatically how the component may at any point ask the speaker questions about a specific message element in order to determine facts that are only important linguistically (for example "person" and "number") or to apply domain-based tests some element, in effect extending the message. In the figure, the message is reduced to its essentials: a composite relation over objects selected by the speaker/expert program (i.e. A(B(C,D), E, F)). We see it broken down within the linguistic component layer by layer starting with the root relation A. B and E have been referred back to the speaker for futher elaboration, while F and then D were realized directly in English (indicated as "..." in the figure). The speaker is accessible continuously, but the timing and the computational context in which it is actually consulted are dictated entirely by the linguistic component.

The speaker has no control over the actions of the linguistic component beyond supplying it with messages; whether the speaker continues to be active while the component is operating is not important to the theory. Whether it should have the ability to interrupt the linguistic component and restart it, perhaps in reaction to what it hears while "listening" to the component's output, is a question we will leave open. There are some eventualities (such as structural ambiguities) that are difficult to foresee when realizing a message via a linguistic component of this design, and there are also potential divisions of effort within the speaker's

planning process which might benefit from a "feedback" design of this sort. Before developing such a design, however, it is critical to have a clear understanding of the kinds of linguistic information that are naturally available at different stages in the production process and of how they relate to the vocabulary of the speaker's planning process—one of the prime concerns of my research. (Clippinger and Brown [Clippinger 1975, 1978 ; R.Brown 1973] developed a model of the production of psychoanalytic discourse that made critical use of such a feedback design, with the result that it was able to produce very natural hesitations and restarts in its monologue.)

4. The Internal Structure of the Linguistic Component

4.1 A cascade of two transducers

As an automaton, the linguistic component is best described as *two cascaded transducers folded together under the command of a single, data-directed controller.* The first transducer goes from the message to a surface structure level linguistic representation of the utterance to be produced—the "working" data structure of the linguistic component—and the second goes from the surface structure produced by the first to English text. (See the sketch in figure 7.)

The "decisions", whose dispositions are so important to this theory, are made almost exclusively by the first transducer; they are the decisions that realize the individual elements of the message through the selection of particular surface structure phrases (or refine existing ones). The second transducer in effect "executes" the decisions of the first by interpreting the surface structure as a program of linguistic actions: printing words, annotating the grammatical context, recording the history of the process, and propagating grammatical constraints.

The bulk of my theory of language production is contained in the characteristics of the surface structure representation and the transducer that produces utterances from it. The transducer from the message to surface structure—conceptually an extention of the speaker—will be less rigorously developed: it is defined chiefly by its relationship to the first transducer—*the controller*—which gates its activities and imposes filters and constraints on its decisions. A complete definition of the first transducer has been developed for use in the computer program and is discussed in detail in [McDonald 1980], however, only certain of its details are critical and the others are expected to be refined and modified as the program is used with real speaker-level planners.

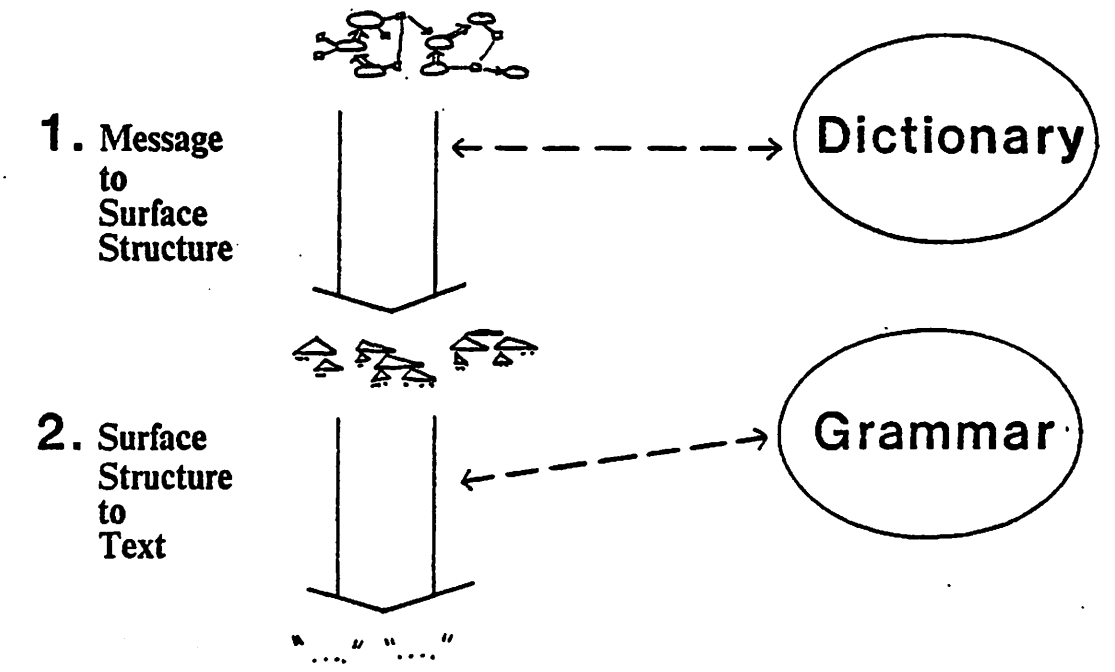


Figure 7 Two Transducers

As inputs to the transducers, both the message and the surface structure are treated as totally ordered sequential streams of data; tokens from the streams are processed one at a time, and are processed only once (i.e. the streams never reverse or loop). The two streams are processed "on-line", which means that the output from the first transducer for one token is completely consumed by the second transducer before the first moves on to its next token. The transducers *per se* are only *interpreters*. They have the ability to follow their input streams and to bind certain predefined variables but little else; their transducing powers derive from two bodies of permanent information, the "dictionary" and the "grammar", to which the transducers will dispatch according to what they find in their input streams. The dictionary associates elements from the message with potential realizing phrases, using a linguistic vocabulary defined by the grammar; the grammar interprets this vocabulary and enforces the constraints and conventional details it specifies. The procedures and schemata in these two "libraries" do all of the real work of the linguistic component; the transducers are responsible for controlling when the libraries are used and for maintaining the linguistic context to which they refer. (The dictionary and grammar consist of a large number of small procedures that are associated with individual tokens that can appear in the data streams (specific message elements, names of grammatical categories, etc.). When a transducer sees one of these tokens, it "dispatches to" the associated procedure (i.e. calls it as a subroutine) and waits until that procedure has finished its execution before going on to the next token.)

The special property of this cascade is that the two transducers have been folded into a single process: the traversal of the surface structure. This procedure can be summarized as follows: The message starts out as the sole constituent of the root node of the surface structure tree; the first transducer then decides which English phrase should realize its dominant element and that phrase, which incorporates at its fringe the next level of message subelements, *replaces* the message in that constituent position. A tree-traversal controller (the second transducer) now takes charge of the process and procedes to traverse this newly constructed surface structure ("*the tree*") following its normal top-down, left-to-right order. As the controller passes over it, the linguistic annotation on the tree triggers dispatches to the procedurally represented grammar and to the dictionary for the realization of the embedded message elements. (The dictionary thus constitutes the real content of the first transducer.) If a fringe constituent is a word, it is printed out as part of the text; if it is a message element, it is realized, replaced in the tree by the new phrase, and the new phrase then traversed as an extension of the surface structure.

The two transducers can be reliably folded together because of a well-formed-ness condition I have imposed on the structure of messages, the "*constraint-preceeds*" stipulation, which dictates that the enumeration order of a message—the position of message elements within the input stream—must be such that any message element that makes reference to other elements in the message must be realized before any of those elements are.

This condition is required because of the stipulation that the generation process must be *indelible* (cf. section 2.3). The theory insures indelibility by designing the tree-walking controller so that it is unable to retrace any part of the surface structure tree after it has passed through it once. When coupled with the locality stipulation, this means that the first transducer is prohibited from arbitrarily scanning the message in search of potentially relevant subelements that denote constraints but must "wait" until those elements are reached in their normal order in the message stream. Consequently if a message includes elements that should be interpreted as constraints on the realization of other elements (for example they might specify discourse focus or pickout attributes that are to be specially contrasted), then those constraining elements should be dealt with first so that their implications can be noted and incorporated into the context of the later decisions. If the process had not been stipulated to be indelible, then we might imagine ordering constraints haphazardly within the message and editing affected parts of the output text by backing up the generator and restarting once a constraint was noticed. Allowing even bounded backup however (as for example with the equivalent of a well-formed substring table) would remove the process from the realm of real-time and would make the "on-line" stipulation impossible to maintain, not to mention requiring a considerably increased memory in order to retain all potentially reopenable states of the process. The "constrain-preceeds" condition and the stipulations of section 2.3 are thus effectively working hypotheses that claim that the appropriate processing trade-off within generation is to supply heavily planned conceptual messages to a relatively unsophisticated but quick and clean linguistic generator, rather than the other way round.

4.2 Representing linguistic context — the tree

In order to understand the two transducers we must understand the data structure that binds them together: the surface structure representation of the utterance under construction known for short as *the tree.* We first describe its format and its relationship to the grammar and the dictionary. We then move on to a sketch of the controller, showing how it traverses the tree and how the tree is used to indicate the proper routines to dispatch to within the grammar and dictionary.

Figure 8 is a diagram illustrating the representation used for the tree. (It is not a snapshot of the tree itself; we will not see one of those until the main example.) Two kinds of structures are indicated: *constituent structure*: defining positions within the tree, how they are connected and how the controller is to traverse them; and *grammatical labels*: defining the properties those positions are intended to have.

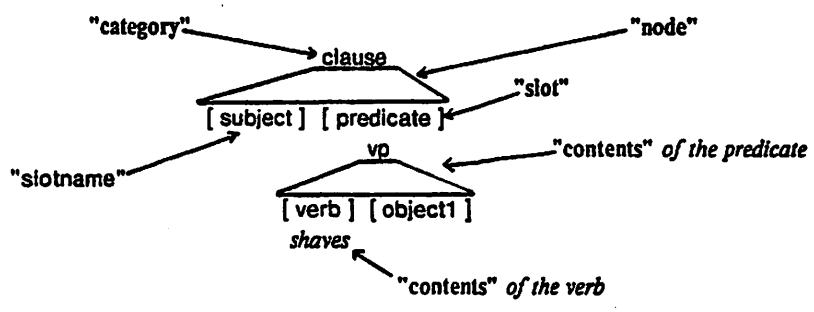


Figure 8 Representing Consitutent Structure: Positions and Labels

The constituent structure is indicated graphically by the pattern of trapezoids and brackets: the trapezoids indicate the "nodes" in the tree, and the brackets indicate the positions of possible constituents within the nodes and are referred to as "slots" or "constituent slots". The actual constituents themselves are the slots' "contents"; for example the node labeled "vp" is the "predicate constituent" (abbreviated "[predicate]") of the "clause node". Besides a node, the contents of a slot may be a word, or a message element, or they may be empty. A subtree from a given node to the fringe of the tree will be referred to as a *phrase.* (Since the tree is always growing through the action of the first transducer replacing message elements with phrases, the notion of the "fringe" of the tree is a dynamic one, constantly changing as the generation process proceeds.) Grammatical labels either label nodes, in which case they may be referred to as "categories" and printed just above the trapezoid; or else they label constituent slots, in which case they are called "slot-names" and printed inside the brackets. A node or slot may have more than one label.

This constituent structure representation is different from most others in the linguistic literature because it explicitly labels the constituent positions rather than just defining them in terms of the relative position of nodes. (This explicit naming of constituents is also done in so-called "relational grammar" [Perlmutter & Postal to appear] and was used in some early phrase structure systems, see [Postal 1963].) The "subject" constituent, for example, could alternatively be defined as the noun phrase node directly under a clause node. Slot-names cannot be dispensed with in the present theory, however, because they are used to carry the grammatical properties of the constituent positions they label. Attempting to use a relative position scheme here would lead either to combinatorially increasing decoding computations as the tree grew in depth or to an undue multiplication of category names; consequently, in this theory the use of explicit slotnames leads to a more natural treatment of grammatical functions.

The constituent structure is only really used by the controller. It defines the path it will take through the tree: a standard, depth-first search pattern as shown in figure 9.

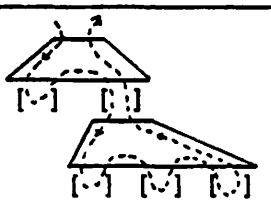


Figure 9 The Path of the Controller Through a Completed Constituent Structure

The importance of this path is in the sequence of grammatical labels that it defines and in the contents of the slots at the tree's fringe. Each label is associated in the grammar with a set of procedures, either of its own or procedures of other labels that are contingent on it; these procedures are referred to as *grammar-routines*. The slotname "subject", for example, has grammar routines of its own (i.e. triggered by the controller when the [subject] is reached, see below) that handle such things as the inversion of subject and verb in questions and the insertion of the function word *"it"* in extraposed clauses such as *"it's easy to be confused by all the terminology"*. The constituent labeled "subject" is looked for specifically by the grammar-routine that performs subject-verb agreement in tensed clauses, and by the morphology routine when it needs to determine whether a pronoun should be in the nominative case.

4.3 The Controller

The algorithm for the controller is the heart of this theory of generation: it *is* the second transducer, interpreting the tree position by position and thereby dictating the order of events within the process, the contextual information available to routines in the dictionary or grammar (i.e. what parts of the tree they can access), and the potential scope of the decisions made by those routines. The algorithm itself is quite simple since all the controller must do is traverse the tree and dispatch to library routines according to the labels on

the positions and the contents of the slots. It is diagramed in figures 10, 11, and 12.

The import of the algorithm lies not in its flowchart which is simple enough, but in the constraints it imposes implicitly on the designer of the grammar and dictionary. *No action can be taken by the linguistic component unless it is specifically selected by this controller at the time and place that the controller dictates.* Thus all actions are local to the controller's position and subject to contextual control No part of the tree is "visible" to the grammar or dictionary except for those parts specifically picked out by the controller's pointers and prearranged pointers positioned by the grammar routines above and behind the controller's position; this means that hierarchical constructions such as embedded clauses or rules with left to right dependencies such as pronominalization or ellipsis can be treated naturally while phenomena with opposite dependencies must be explicitly planned for or they will be missed. Similarly, since only a single position in the tree is seen at a time (in contrast with the multi-position buffers of natural language parsing systems such as [Marcus 1980]), phenomena that can only be seen by processes with a distributed view of a constituent structure—such as structural ambiguities—cannot be easily appreciated by this system and will typically go uncorrected. This controller is thus the embodiment of the hypothesis that only hierarchical and sequential dependencies can be appreciated during immediate speech; all others being either expressly anticipated and planned for ahead of time or left to a *post hoc* monitor to detect and compensate for later.

MUMBLE (message)

argument: a message
return value: none

The initial tree: **root-node**

[root-constituent]

START

Initialize the environment:

```
current-grammatical-filters <= 'empty
discourse-history           <= 'empty
<all grammatical-variables> <= 'undefined
```

Initialize Controller Variables:

```
current-node     <= 'root-node
current-slot     <= 'root-constituent
current-contents <= (Make-elmt-instance message)
```

A Start the Controller at "Dispatch on current-contents"

Figure 10 The Controller: Initialization

Initialization When a new message is passed to the linguistic component, it becomes the contents of the constant slot "root-constituent". If at that time the component has finished processing any earlier messages then it will have returned to that position at the top of the tree and the message's processing will start immediately; alternatively should an earlier message still be in progress it will not be disrupted and nothing will happen to the new one until the old one is finished and the controller completed its traversal back to the root. The initial state of the controller is shown in figure 10.

The Block-level Organization of The Controller The controller decomposes into three recursive procedures named *Process-node*, *Process-slot*, and *Dispatch*, whose definitions are given in the next two figures. They are threaded in that same order: Process-node calls Process-slot on each of its immediate

consituents, and Process-slot in turn calls Dispatch on the content of the slot presently being processed. The recursive structure of the tree is matched in the controller algorithm by the recursive call to Process-node from within Dispatch.

The algorithm differs from the standard recursive descent algorithm for traversing a tree only in its "realize and replace" step within Dispatch: This step has the effect of dynamically extending the tree even while the controller is traversing it. The extension stops when a phrase is selected that has no further message elements embedded at its fringe. The dynamic extension of the tree as is the key to the *progressive refinement* technique that characterizes this theory: the embedded message elements in effect constitute "delayed decisions" that are not taken up until all of the prior decisions that might effect them have been made and their constraints established (all such dependencies being, by hypothesis, associated with positions in the surface structure above and behind the embedded element). This technique is akin to the technique of *delayed binding* that is used in the processing of some programming languages.

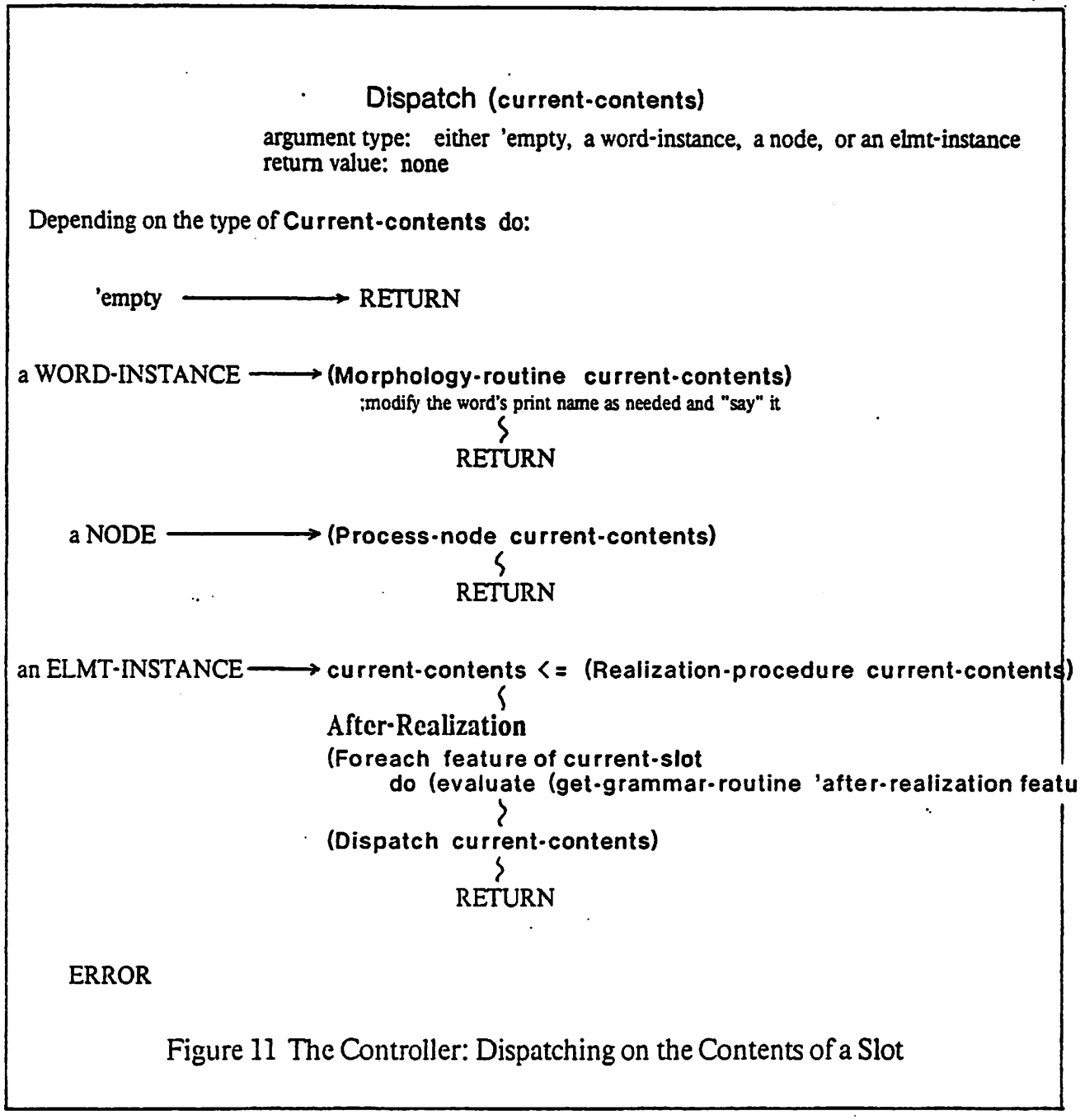


Figure 11 The Controller: Dispatching on the Contents of a Slot

Gating the First Transducer The first transducer is taken up exclusively in the dispatch to the function named "realize". This function contains the procedures and heuristics that are common to all realization decisions: i.e. criteria for pronominalization, alternate subsequent reference forms such as *"one"* or *"such"*, and "gap-creation" for WH-movement; if none of these apply, it dispatches to the message element's dictionary entry and the standard entry interpreter sketched later in the paper (for a complete specification of

the realization process, see [McDonald 1980]).

Every message element that is embedded in the tree must eventually pass through this step of the controller, and then and only then will its English realization be decided on. On return from the function Realize, the selected node, word, or subelement is knit into the tree in place of the original element, at which point the controller loops around and repeats the dispatch on the new contents of the slot.

A Last-stage, Morphological Process When an English word is found as the contents of a slot, it passed to a procedure named the *Morphology routine* for any required specialization and from there to the output stream. The Morphology routine does not have much work to do in English: it is responsible for the case of pronouns, for plural forms, verb conjugation, contractions, and the possessive. It bases its decisions on properties it associates with the slot-names of the slot that contains the word (e.g. "subject" is understood as forcing the nominative case) and on inherited attachments to the constituent structure marking such "extra-constituent" information as tense, aspect, and negation. It is the routine within the generator with the clearest representation of the notion "next word", and as a result is responsible for grammatical phenomena dependent on sucessive linear position such as the verbal auxiliary.

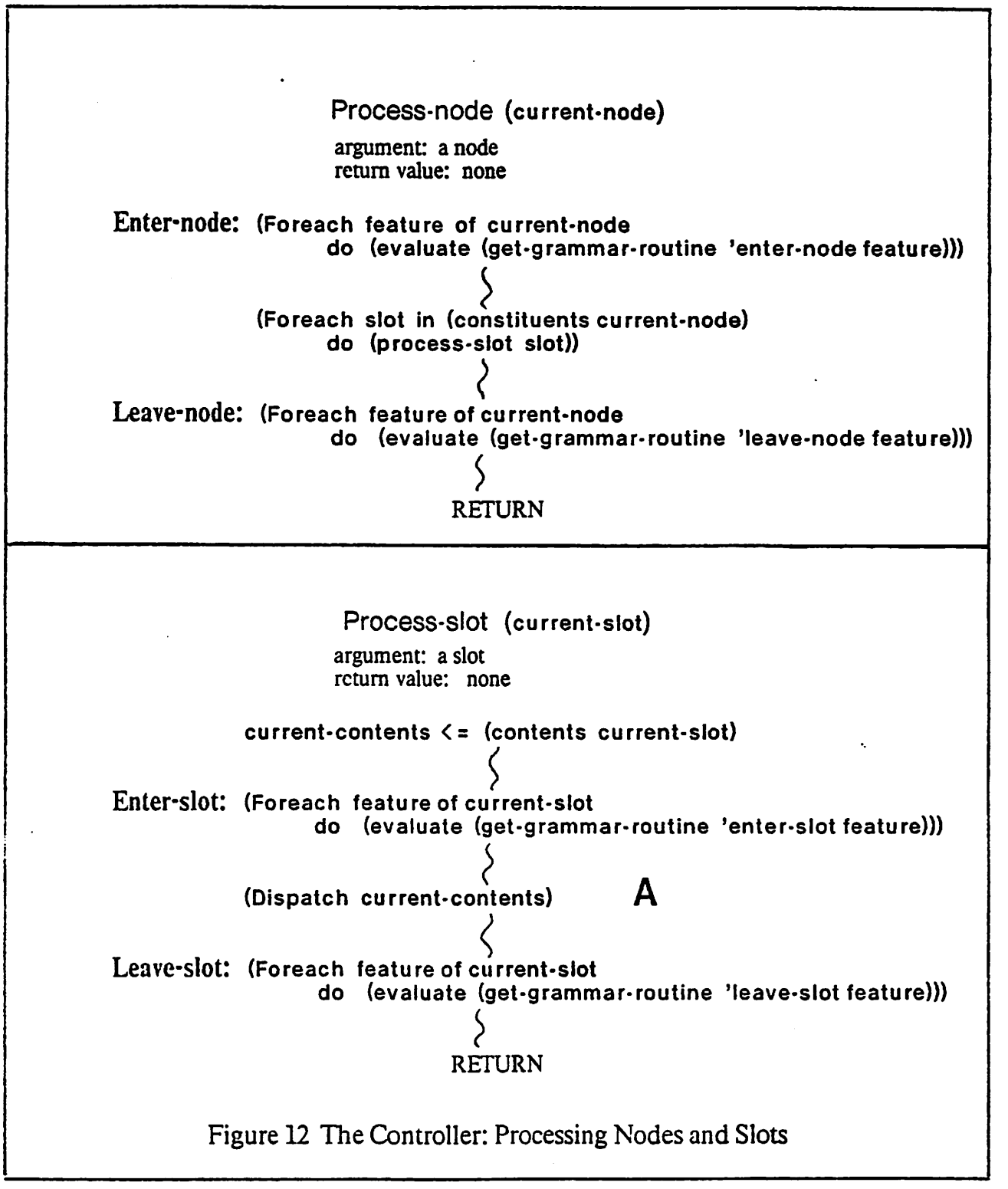


Figure 12 The Controller: Processing Nodes and Slots

Associating Grammar-routines With Constituent Structure Labels The two procedures Process-node and Process-slot are the primary place where the library of active procedures that constitutes the active aspect of the grammar is used. These procedures are referred to as *grammar-routines* and are associated with specific grammatical labels (also referred to as *features*). Grammar-routines are further specified by the point in the controller's algorithm where they are to be executed, marked in the flowcharts in **bold** type. they are five generic events in the traversal of a tree All of the active parts of the generator's English grammar are associated with the labels attached to the nodes and slots of the surface structure. These points correspond to five generic "events" in the traversal of the tree: entering or leaving a node, entering or leaving a slot, and just after a message element has been realized but before the realizing phrase has been knit into the tree. As indicated in figure 12, when one of these events is reached, each of the labels associated with the current node or slot is checked for a grammar-routine of that event type, which if found is immediately executed.

Grammar-routines may perform any of the following actions:

(1) Add function words directly into the output text stream;

(2) Set or reset reference pointers ("grammar variables") to immediately accessible parts of the tree for the maintainance of grammatical context (see below);

(3) Make specifically constrained "edits" to the constituent structure they immediately dominate so as to impl nent locally triggered phenomena such as heavy phrase shift or conjunction reduction;

(4) Make local "grammatical decisions" that are not required by the speaker's message but are necessary grammatically such as the selection of complementizers.

The current context The rules of grammar embedded within the grammar routines are couched in a vocabulary that is always interpreted with respect to the current position of the controller in the tree. This position is defined in terms of the values of three variables: *current-node, current-slot,* and *current-contents,* which are set and reset as the controller moves. Grammatically important facts about the tree—the vocabulary of the grammar rules—are represented in terms of a set of variables that are bound locally in the tree but have their values set and reset by of specific grammar-routines. The three variables above are referred to as *controller-variables,* and a second, open-ended set are termed *grammar-variables.* In addition to these variables, the controller maintains a *discourse history,* consisting of records of all important events that have occurred, including the realization of every message element instance, every selected choice, and every decision brought about by the grammar.

In summary, the current context of the linguistic component can be viewed as a four dimensional array consisting of (1) the name of the controller event or subroutine presently being executed, (2) the values of the three controller-variables, (3) the values of the grammar-variables, and (4) the records of the discourse history. This representation of the context will be used in the diagrams of the main example.

5. An Example

This example should serve two purposes: first, to put flesh on the apparatus of the linguistic component just discussed by showing how it acts as a system; and second, to illustrate some of the sorts of linguistic analysis that one is lead to as a scientist working in terms of this theory of language generation. From the point of view of conventional, competence-based linguistics some of the analyses that will be sketched may seem unusual or even bizzare; this is perhaps to be expected since the need to smoothly interact with an independent, non-linguistically based process (the speaker/expert program) has imposed its own mark on the analyses everywhere from the timing of decisions to the details of the surface constituent structure.

This example is drawn from the logic domain described in section 1.2. We will look at the generation of the last part of the "barber proof": initially in considerable detail in order to demonstrate how the controller interacts with the selected surface structure, and then at a coarser level of detail so as to concentrate on the analyses and the motives behind them. The example will actually be only the last two lines of the proof, but we will put those lines in context first by sketching the events up to that point.

Generation in the logic domain is an example of "direct translation". There is no planning component; instead, messages are constituted directly from the regular data structures of the domain, the lines of the proof. This is the characteristic pattern of direct translation systems (for example [Swartout 1977; Shortliffe 1976]), and it the source of their convenience—side-stepping an elaborate planner by taking advantage of the organization already in the domain's native data structures, as well as of their limitations—they lock the generator into a single level of abstraction and invariably leave many conceptual connections implicit. By translating first into a linguistic representation and then applying general grammatical rules and usage heuristics, we are able to generate a smoother, more natural text than earlier generators that translated directly into word strings; however, the overall form and content of the text remain in the mold set by the input proof. It is safe to say that the direct translation technique is pushed here to the limits of its fluency; further improvements will only come with the addition of a planner with a knowledge-base of rhetorical heuristics.

The "message" that started the generator off was the seven lines of the proof in sequence. Figure 13 is a snapshot of the tree ju . after this message was received and distributed into the slots of a simple paragraph; note that the order of the lines has been preserved in the left-to-right sequence of the slots. The formulas have been abbreviated to just the names of their lines.

line1: premise
$\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y)))$
line2: existential instantiation (1)
$barber(g) \wedge \forall y(shaves(g,y) \leftrightarrow \neg shaves(y,y))$
line3: tautology (2)
$\forall y\ shaves(g,y) \leftrightarrow \neg shaves(y,y)$
line4: universal instantiation (3)
$shaves(g,g) \leftrightarrow \neg shaves(g,g)$
line5: tautology (4)
$shaves(g,g) \wedge \neg shaves(g,g)$
line6: conditionalization (5,1)
$\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y)))$
$\rightarrow (shaves(g,g) \wedge \neg shaves(g,g))$
line7: reductio-ad-absurdum (6)
$\neg\exists x\,(barber(x) \wedge \forall y(shaves(x,y) \leftrightarrow \neg shaves(y,y)))$

Becomes:

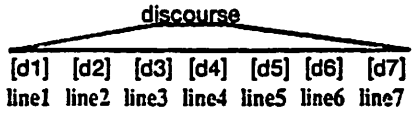


Figure 13 Message and Initial Snapshot

The fixed traversal pattern of the controller dictates that the text will be produced incrementally following the sequence of the lines. This guarenteed conventional sequence provides the basis for a chronological discourse context: The text for the first line will have been selected and produced before that of the second line is begun, the second before the third, and so on. On this basis, a model of what the listener will have heard can be infered, and, coupled with a (very simple) model of what inferences the listener will make or can be lead through, will give us some justification for extending the context-free interpretation of the lines of the proof to an interpretation in terms of the roles the lines play in a conventional proof technique. Thus while there are seven lines in the proof and seven sentences in the text, there is by no means a one-to-one mapping: The first line of the proof, the premise, is rendered as an imperative to the listener, setting the form of the argument as a proof by contradiction. The second line instantiates the variable. Logically its formula is a restatement of the body of the existential formula from line one with a constant substituted for the body. It is realized in the text however only in terms of its role in the proof, *"Call him Giuseppi"*, the formula itself being appreciated as redundant.

The third line does not appear in the text *per se* at all since it is an obvious conclusion from what was known so far. The fourth line, on the other hand, has been expanded into a three sentence "mini-argument" because of its importance to the proof and because its logic may not be obvious. The fifth line, the derivation

Assume that there is some barber who shaves everyone who doesn't shave himself (and no one else). Call him Giuseppe. Now, anyone who doesn't shave himself would be shaved by Giuseppe. This would include Giuseppe himself. That is, he would shave himself, if and only if he did not shave himself, which is a contradiction. This means that the assumption leads to a contradiction. Therefore, it is false, there is no such barber.

Figure 14 The "Barber" Proof

of the contradiction, is interpreted for its conventional role, i.e. announcing the derivation of the contradiction. In the text it is adjoined to the previous sentence as a relative clause—a kind of "renaming" speech-act.

We see the changes that these realization decisions have made in the tree in figure 15. Only the top nodes of the sentences are shown. The controller is now positioned at slot "d6", and the two final lines of the proof remain. In looking at the generation of those lines, we will be begin with very cursory descriptions of the first few lines to establish the basic pattern, then move to very detailed snapshots of the controller and the tree for several decisions that involve straight-forward analyses, and then back away from the detail during the last line to highlight the special kind of reasoning that generation can entail. For further examples and a thorough discussion of the analyses, see [McDonald 1980].

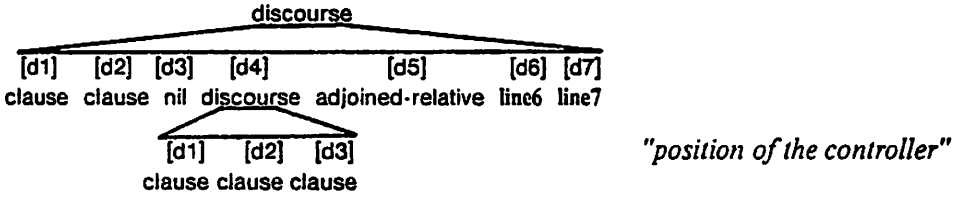


Figure 15 The Tree after Line 5

5.1 Recursive Descent Through the Formula

The logical decomposition of line6 begins with the relation between the inference rule "conditionalization" and the formula it derives (abbreviated "formula89 — conj101"); thus the generator must do the same. The dictionary entry for conditionalization must select a phrase that will convey how the formula is related to the rest of the proof before it, and then the formula will be realized in the context of that phrase. By default, we have the inference rule realized as a bridging phrase stating the conection, *"this means that..."*, which embeds the formula as a complement. The knowledge-base of the domain would not motivate

anything more elaborate without appeal to a richly annotated, self-conscious theorem-prover, and one was not available. The controller traverses this fixed phrase "saying" the subject and verb. The complementizer *"that"* is produced by a grammar-routine associated with the label "complement" rather than from its own slot in the tree because of a design hypothesis that says that slots should be reserved for items derived directly from the message; function words are by hypothesis a part of the linguistic background just like the annotation on the tree.

The next step in the descent is the major connective of the formula, the implication. Implications can take many forms in English, but the most direct is the subject-predicate relation selected here. The verb *"leads to"* is specific to this conventional use of the contradiction. The snapshot in figure 16 show the tree from [d6] down at the point after the clause realizing the implication has been put in place. The four parts of the controller's state are given explicitly.

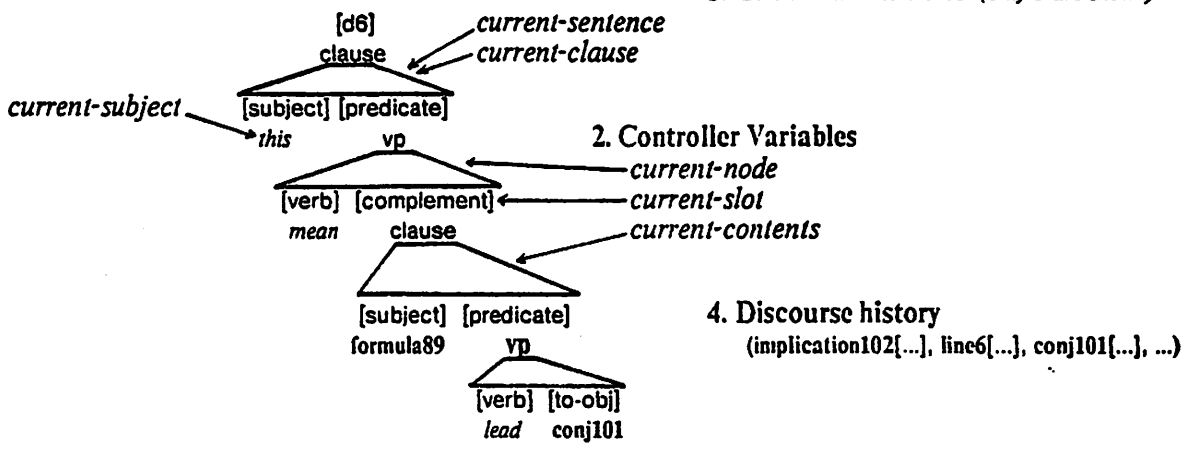


Figure 16 Snapshot of the Controller's State

5.2 Stepping The Controller Through the Tree

With the controller in its "Dispatch" state, the next step from the position of the snapshot is a recursive call to Process-node (refer to the earlier flowcharts). The controller variable "current-node" is reassigned to the new node labeled "clause", and we execute any grammar-routines associated with the label "clause" and the controller-event "enter-node". There are presently two of these: one for assigning the current-sentence, which no longer applies, and one that recursively reassigns the grammar-variable "current-clause", which does. From here the controller moves to Process-slot, reassigning "current-slot" to the first of the clause's

constituent slots, the "subject", and "current-contents" to formula89. One grammar-routine applies here to assign the grammar variable "current-subject" to the current-contents. (Working in recursive environments such this clause requires care in the timing of assignments. By "delaying" the updating of the pointer to the subject until now, we have retained access to the higher subject where it was needed, e.g., for potential applications of equivalent-np-deletion or conjunction reduction at the level of the clause.)

From Process-slot, the controller calls Dispatch and selects the "msg-elmt" case. The function Realize will now control the selection of a realizing phrase for formula89 which will then become the current-contents and the controller will loop through Dispatch again. The realization of formula89 involves appreciating its redescription as an object with a special role in the proof, i.e. *"the assumption"*. Its dictionary entry is considerably more involved than average; consequently rather than look at the interpretation of that entry, we will digress here to consider a more "normal" entry and how it fits into Realize. The redescription technique itself will be described later.

5.3 The Realization Process

Figure 17 is a high-level flowchart of the function Realize: It divides into two paths depending on whether this is the first instance of the message element to appear in the tree, in which case we go directly to its dictionary entry, or whether this is a subsequent reference (as with formula89), in which case we apply various heuristics to determine if it should be realized as a pronoun or some other form of "subsequent reference". (Summarizing an entire formula with the phrase *"the assumption"* is a form of subsequent reference.

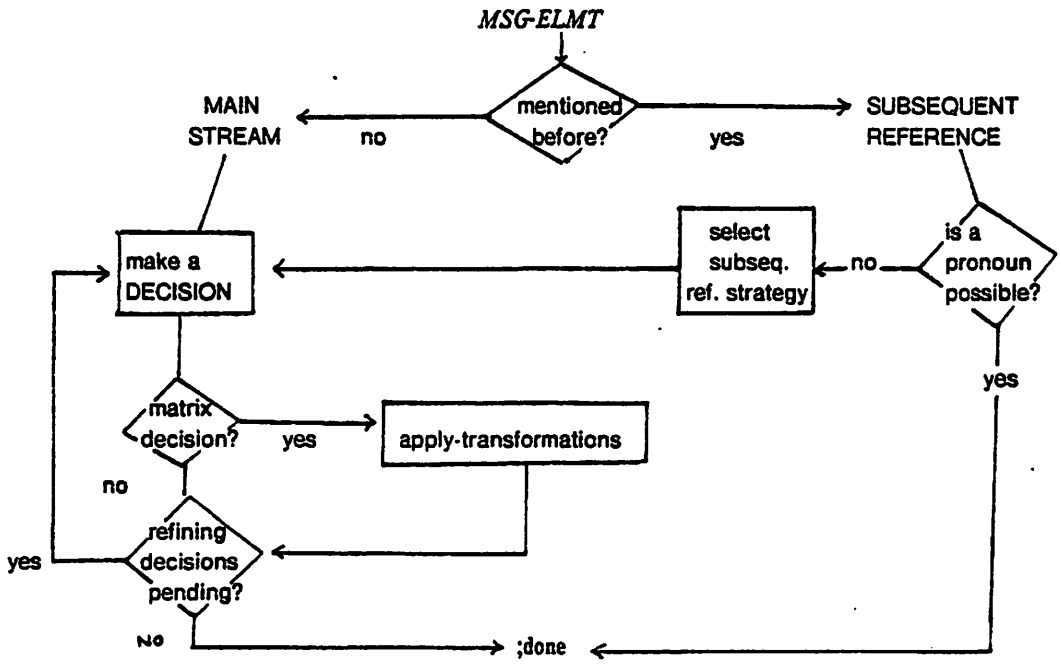


Figure 17 flowchart of the realization procedure

Every entry has a "matrix" decision, the one that determines what category of phrase will be used, e.g. noun phrase or clause, and may have an arbitrary number of other "refining" decisions that can add additional features to the phrase or add optional constituents.

A dictionary entry consists of a set of possible "choices" and a set of "decision-rules" to pick between them: A *choice* is a symbolic specification of phrases, words, or subelements of the element being realized; A *decision-rule* has two parts: one, a list of predicates that may examine both the linguistic context and the context of the speaker, and two, the choice that should be selected if those predicates are true. The bulk of the realization process consists of interpreting the decision-rules to select a choice, then possibly going through further sets of decision-rules to see if the grammatical or rhetorical context dictates that the choice should be transformed. Extensions to the tree occur when the selected choice specifies a phrase.

The vocabulary of the specification comes from the permanent knowledge base in the grammar, part of which is a listing of all of the legitimate categories in the language and for each category, of the legitimate sequences of slotnames that it can dominate. These listings are organized in terms of "constituent schemas". ("Schemas" in the sense that they will be used as templates for the construction of "instances" of those category configurations in the tree.) Every choice has (at least) three parts: (1) a "phrase-schema" that defines a tree of constituent-schemas possibly augmented by additional labels and by specific words from the English

vocabulary; (2) a list of formal parameters that will be used to pick out subelements of the message element being realized; and (3) a mapping from parameters to slots at the fringe of the specified phrase.

Figure 18 lists the entry, choice, and grammar that are required to realize the logical predicate *shaves*.

Message Element shaves(x,y)

Dictionary Entry

```
(define-entry shaves-entry (shaver shavee)
    (matrix
        default (clause-direct-object shaver "shave" shavee)))
```

the Choice

```
(define-choice clause-direct-object
    parameters (subj verb object)
    phrase (basic-clause ()
                predicate (vp-obj1 () ))
    map ((subj . (subject))
        (verb . (predicate verb))
        (object . (predicate object1))) )
```

Constituent-schemas

```
(define-schema basic-clause
    categories (clause)
    slots (subject predicate))

(define-schema vp-obj1
    categories (vp)
    slots (verb object1))
```

WHICH PRODUCE:

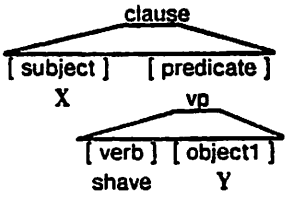


Figure 18 Entry, Choice, and grammar for the relation 'shaves'

The dictionary entry *shaves-entry* will be the one to perform the realization. It decomposes the original message element into two subelements, the variables X and Y plus the verb *"shave"*, and binds them to three local variables for ease of manipulation. Shaves-entry has only one choice, which it has marked as its default. As it includes no decision-rules, this default choice will always be taken. The choice, *clause-direct-object* (named for the kind of consituent structure it builds), is given in the figure just below the entry. It uses two constituent schema, *basic-clause* and *vp-obj1*, to define a two-level phrase-schema (shown instantiated at the bottom of the figure) whose *verb* constituent has been filled in but other constituents left vacant in anticipation of being filled by message elements selected by the entries that select clause-direct-object. Because this choice will inevitably be used with many different entries, its mapping is given in terms of its own formal parameters (i.e. *subj*, *verb*, and *object*), which are then bound to the values of the local variables of

the entry when the choice is taken. To produce new constituent structure from the choice, its phrase-schema must be instantiated and the mapping applied to to fill its leaves with the message elements the entry has selected.

5.4 Continuing through the tree

Once the phrase for formula89 has been instantiated and knit into the tree, the controller as before recurses on Process-node and begins to traverse the new noun phrase. Nothing new happens within the phrase, so we will move on to the next snapshot, taken after the controller has finished with the subject and moved down and through the slot "to-obj" to the object constituent.

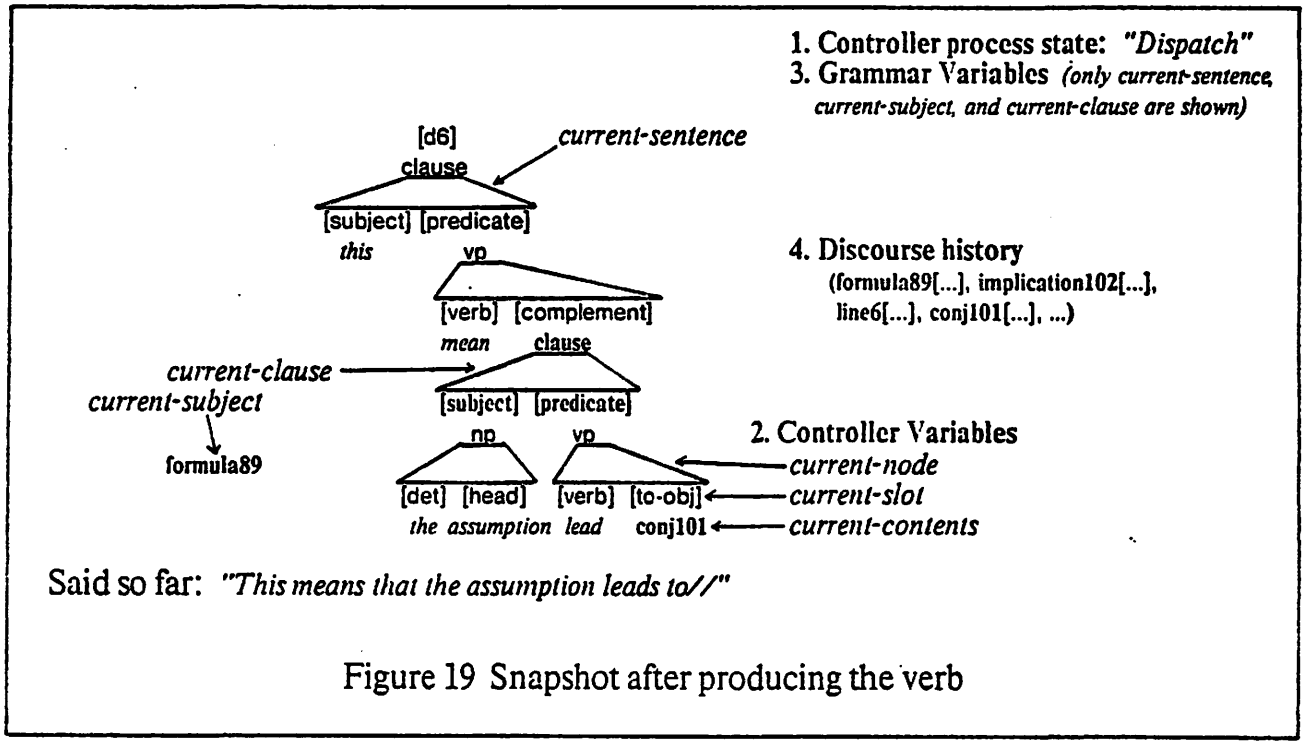


Figure 19 Snapshot after producing the verb

Notice that we have not replaced the verb with its third person singular form even though that is the form that appears in the output stream. By design, we have decided that the level of representation exibited in the tree should be the level needed by the library routines that reference it; we will not gratuitiously "update" its contents or add labels if they are not going to do further work in the grammar. The correct morphological form of the verb was needed only for the output text and so was not constructed until the morphology routine was passed the word on its way to the text stream. Later grammatical references to the verb are going to be concerned with its grammatical properties rather than its morphological ones (for example whether it can take complements, and if so whether they are subject or object controlled) and these properties are by convention associated with the root form of the verb which is the one in place in the verb slot. The preposition *"to"* was introduced by a grammar-routine attached to the label "to-obj" rather than

having its own slot for the same reason: we expect no other part of the generator, dictionary or grammar, to need to know about the presence of that preposition so we make our expectation concrete by having the preposition completely "invisible" within a grammar routine rather than occupying a slot where it could be noticed.

Redescription according to function The last significant operation before moving on to line7 is the realization of conj101, the formula **shaves(g,g) ∧ ¬shaves(g,g)**, as the English phrase *"a contradiction"*. This is of course not a literal rendering of the formula; that would have been *"He shaves himself and he doesn't shave himself"*. Instead it is a rendering of the *conventional role* that the formula played in the proof at that point, i.e. an indication that a contradiction had been derived. This ability to realize expressions in terms of their functional redescriptions was also used in the realization of formula89 as *"the assumption"* at the beginning of the sentence.

Redescription is a way of seeing the same concept or operation at multiple levels simultaniously depending on one's intent, and has become an important part of the representational "repertoire" of modern expert systems, where it is used in plan recognition and in defining levels of abstraction (see particularly [Mark 1981]). Intuitively, redescription is associated with particular turns of phrase in English such as appositives (as in the last sentence of this example) or some noun-noun combinations (e.g. *"the role pobj"*); consequently, it is useful to make specific arrangements for it within the linguistics component.

Ordinarily, redescription would be an operation at a conceptual level rather than a linguistic one, and we would expect it to be explicitly indicated in the message; however, since the present microspeaker has no real conceptual knowledge of logic and starts with only the bare formulas of the proof, we must compensate by performing the redescription locally within the dictionary. The relevant parts of the dictionary are the entries for the inference rules, these being where the microspeaker's tacit knowledge about the structure of proofs resides. The redescriptions of the individual formulas are deduced as the entries are interpreted and stored within the linguistics component on a special association list: the entry for a Premis, for example, notes that the formula on its line serves the function of being the assumption of the proof; and the entry for tautologies (which is actually a clearing-house for an entire set of logical manipulations) notes that any derived line of the form **A ∧ ¬A** is serving to mark the derivation of a contradiction at that point in the proof.

The access function that associates formulas with their entries includes a special check for redescriptions and passes every redescribed formula to a common *meta-entry* for its realization. (A meta-entry chooses between other entries rather than between English constructions.) This entry knows how to use the special redescription phrases, and has access both to the literal renderings and to the functional-level renderings, which it combines according to the context (see discussion below in conjunction with the realization of neg103).

5.5 Delaying Decisions

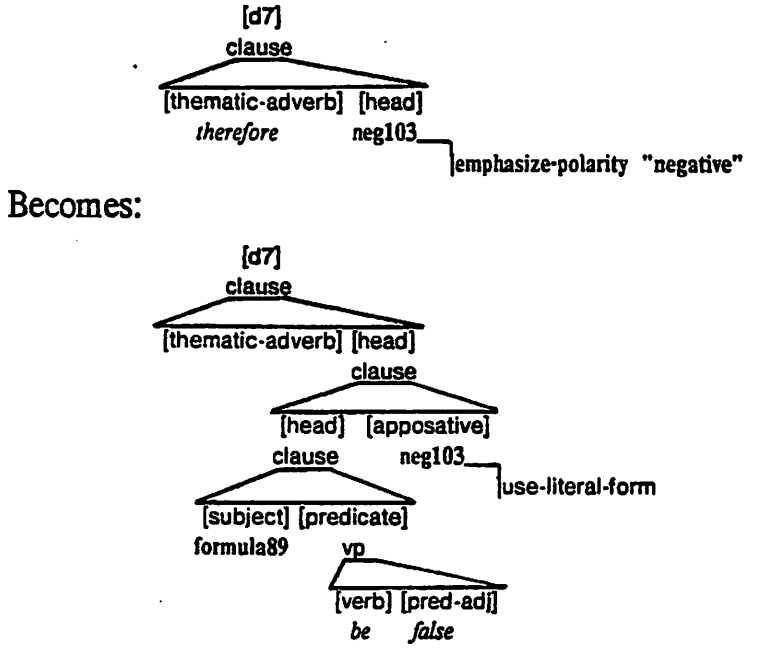
One problem that can arise with the direct-translation technique is that while the formal structure of the data used in a message (here the predicate calculus) may be convenient within the domain, it can be at odds with what would be convenient for the generator. Negation103, the last line of the proof, is a case in point.

$$\neg\exists x\,(\text{barber}(x) \wedge \forall y(\text{shaves}(x,y) \leftrightarrow \neg\text{shaves}(y,y)))$$

Linguistically, the principle contribution to the content of any text created from negation103 will come from the "shaves" relations, yet these are the most deeply embedded in the formula. If we follow the formula's natural decompostion order, five logical operators will have to be passed through before these content relations are reached. Given the indelibility stipulation, the realization decisions for those operators must be made "on the way down" as it were, but how can this be done if those decisions are contingent on linguistic details of how the content relations can be realized—details that will not be determined until those relations are actually reached by the controller.

This problem is solved by delaying the affected decisions. We postpone them until the information on which they depend has been determined, implementing only those decisions that can be made independently and attaching annotations to the tree indicating that the remaining decisions are still pending. Consider the first two realization decisions in line7: The inference rule, *reductio-ad-absurdum*, is realized like the previous conditionalization rule by picking a bridging adverb that will convey the fact that the line is the conclusion (e.g. *"therefore"*) and then embedding the line's formula in that linguistic context. However, because this is a proof by contradiction, we know by convention that this final formula will be a copy of the premis but with opposite polarity. We should somehow emphasize this polarity in the text, but from this vantage point in the process we do not yet know what linguistic mechanism should be used (e.g. an explicit "do" or an emphatic "not"). We must thus delay the decision until we know more, which means that we add an annotation to the formula we embed, expecting the annotation to be recognized by later routines that will be active when the needed information is known.

As it happens, the dictionary entry of the very next operator in the decomposition, the negation, has an alternative among its choices that we have determined in designing the grammar will serve to emphasize negative polarity, i.e. *"<body of the negation> is false"*. The negation entry is allowed to select this choice if the body of the negation, the premis line formula89, can be expressed as a simple noun phrase (nominalized clauses are disallowed); that is the case here, since formula89 was just referred to in the last sentence as *"the assumption"* and that nominal form will carry over. If only a clausal realization had been possible, then the negation decision would have been delayed as well.




```
(define-entry negation-entry (neg)
    variables ((body (body neg)))
    (matrix
        ((equal (get-annotation neg 'emphasize-polarity) 'negative)  ;decision-rule
         (will-be body 'np)
         (X-is-false body))  ;choice to select

        ((will-be body 'np)  ;i.e. do it whenever possible because it's more fluent
         (X-is-false body))

        default (Mark-X-negative body)))
```

Figure 20 Delayed Decisions

Expressions like *"The assumption is false"* lend themselves to appositive phrases that expand on what was summarized in the subject. This is a fact of the grammar that can be implemented in a generator in terms of a transformation that we associate with this expression. The transformation acts independently once triggered by the use of the expression and examines the subject to see if an appositive would be appropriate. The heuristic used here is a very simple and arbitrary one: redescriptions such as *"an assumption"* or *"a contradiction"* will draw apposatives if their long forms have not been mentioned within three sentences. The same heuristic applies to pronominalization decisions and is intended to reflect when a reference has faded in memory—research on discourse structure should lead a more principled criterion. The fact that the apposative has been planned at the level of the clause inhibits it from appearing redundantly with the noun

phrase; that is, the local decision that would have produced *"The assumption that there is no such barber is false, there is no such barber"* is filtered out by the presence of the higher apposative in the tree.

5.6 Interaction Between Decisions

Traversing the tree below the clause that realized the first instance of negation103 is a simple matter. The embedded formula in the subject is pronominalized because of its proximity to its last instance and the fact that both instances were in subject position (in effect a "poor man's" rule of discourse focus). The pronoun, verb, and adjective are then passed to the output stream as the controller moves through their constituent positions.

At the position of the apposative, the negation entry this time passes its decision down to a later process since its body has been specially annotated because of its apposative function so as to block the redescription of formula89 as a noun phrase. We go then to the next level of neg103, the existential quantifier, where we have two choices: either to pass the realization of the quantifier down to appear as the determiner in the realization of the variable (as in *"Someone shaves everyone who doesn't shave himself"*) or to use the special existential construction *"There is"*. As one might imagine, this decision is designed to be sensitive to the pending decision on the negation, and we select the special construction since the negation would preempt the determiner and make the other alternative ineffective.

In English, clauses with the existential *"there"* are grammatically unusual because the verb agrees in number with the object rather than the subject. This is handled here via the same mechanism as presently used in transformational grammars: The word *"there"* is taken to be a lexically filled "trace" pointing to the logical subject of the clause, such that when the grammar routine that implements "subject-verb" agreement refers to the variable "current-subject" it is passed transparently to the object instead. (The reference is not actually to the "object"—it cannot be, since the position of the object has not yet been reached by the controller and thus its contents cannot be known. Instead the trace points to the message element that would have been the subject if *"there"* had not been used, and the transformation that introduced the word *"there"* redirected that element to the object position.)

5.7 Realizing Message Elements in terms of their Roles

The conjunction that is now the object constituent of the *"there"* clause is yet another conventional expression within the proof.

$$\mathbf{barber(x)} \wedge \forall \mathbf{y(shaves(x,y)} \leftrightarrow \neg \mathbf{shaves(y,y)}$$

The predication "barber(x)" is a restriction on the variable **X**, which, in other logical notations, would have appeared in other places in the expression. The actual "content" of the conjunct is just the universally quantified formula. If we knew nothing else about this conjunction, we would be forced to realize it literally, as in *"Someone is a barber and he shaves everyone who doesn't shave himself"*. However, if we make the conventional structure of the conjunction apparent to the linguistic component, we can be much more fluent.

By labeling the predication as a "description" and the formula as a "proposition" both predicatable on the variable **X**, we can take advantage of a general purpose dictionary entry for that combination.

As shown in figure 21, à description and a proposition can be combined in several ways according to what is needed (i.e. which of the two elements is more important, which order is more important, etc.). In this case since the conjunction is acting as an object, the combination where the two are set up a modifiers in a noun phrase referring to the variable is the most appropriate, and that is the phrase that is built and replaces conj88 in the tree.

Combinations of a description and a proposition predicated of the same object.

proposition[object]
description
Someone who is a barber shaves everyone who doesn't shave himself.

description[object]
proposition
Someone who shaves everyone who doesn't shave himself is a barber.

[$_{np}$ object]
modifier
proposition
A barber who shaves everyone who doesn't shave himself.

Figure 21

This particular conjunction has of course appeared before in the first line of the proof. We are therefore dealing with a subsequent reference and the heuristics in that section of the Realize function apply. Because of the distance of the original instance from the present position, the conjunction should not be pronominalized, however there are of course "intermediate" subsequent reference strategies. One of these, particularly appropriate to the style of a mathematical proof, is the word *"such"*; this word can "pronominalize" the modifying phrases of the reference, leaving its head and determiner.

6. Contributions and Limitations

6.1 Specific Contributions of This Research

The computer program developed in this research (see [McDonald 1981]) is the most linguistically competent natural language production program that has been reported to date. This is due primarily to the advances in the computational theory of production reported here and in [McDonald 1980] which have simplified the process of representing linguistic rules and usage-heuristics. In particular:

(1) This is the first theory to be specifically designed for use with source programs that use different representational systems.[5]

(2) This is the first theory to be grounded on pycholinguistically plausible hypotheses embodied in a processor of limited computational power; relevant hypotheses include: the left to right refinement and production of text,[6] linguistically motivated limitations on the examinable buffer, indelible decisions, and a structural distinction in the treatment of function words versus content words.

(3) Production is driven directly by the message to be expressed, not by the hierarchical structure of the grammar. This is more efficient, and facilitates the conceptualization of messages as descriptions of goals to be achieved by the text.

(4) The linguistic structure of the text being produced is explicitly represented.[7] Grammatical rules can be implemented directly as manipulations of linguistic descriptions, thereby gaining generality and perspicuity. Details of the structure of produced and planned text may be referenced directly and used as the basis of usage decisions.

(5) The possible realizations of each element of a message are explicitly represented and are available for inspection or special-case manipulation.

6.2 Relation to previous A.I. work on natural language generation

Virtually all of the earlier work on language generation by people in A.I. including that of this author and most of that done by psychologists shares a common view of the process: an expert-program/speaker with no linguistic knowledge or motivations begins the process by deciding—in its own terms—what will be talked about. Instead, the differences between the various proposals concern the kind of device that is to take such a "message" and to produce a natural language text from it through application of grammatical knowledge (encoded in some form) and the use of some kind of "dictionary" to interpret the speaker's message.

---

5. The ATN-based generator originally developed by Simmons and Slochum [1972] and later adopted by Goldman [1974] has been used with many different programs: [Reisbeck 1974; Lehnert 1977 ; Yale A.I. Group 1976; Meehan 1976]; however, all of these employed the same representational system: conceptual dependency [Schank 1976].
6. Gerard Kempen [Kempen 1977] writes that production should be incremental and left to right, however, his program as described in [Kempen & Hoenkamp 1980] while realizing clauses sequentially, refines the constituents of each clause in parallel.
7. This was true also of the German-to-English translation program of Gretchen Brown [1972], and locally true in the systemic grammar used by Anthony Davey [1974].

Two other perspectives have been taken (see [Mann et al. to appear]): one school can be termed *grammar-controlled linearization and translation* [Simmons & Slochum 1972; Goldman 1974; Shapiro 1975]; another, larger though less linguistically sophisticated school can be termed *production directly from program data* [Swartout 1977; Chester 1976]. (Two other important systems, [Clippinger 1978] and [Davey 1974], fall into neither of these categories as they both employ extensive grammars and vest control with non-grammatical processes; unfortunately, neither has been futher developed.)

The *grammar-controlled* school vests total control of the process in a topdown generative grammar, typically given as an augmented transition net ("ATN"). This grammar hypothesizes a way in which the message might be realized, and then tests the message to see if that way is feasible. It constructs the hypothesized text if the test suceeds; otherwise it backs up and considers the next grammatically possible realization. Texts are produced as a side-effect of traversing the ATN. Compared with using the message decomposition itself to control the process, this technique is inefficient at best, and at worst, allows the possibility of producing totally confused text should the ATN ever backup over an arc-path that produced words (i.e. it would start repeating itself without regard for context). Historically it is the case that none of these systems has ever had occasion to backup; we conjecture that the reason for this is that the space of possible message configurations dealt with by these sytems is relatively small, making it possible to directly encode the space on the arcs of the ATN grammar as tests for all of the possible contingencies. We predict that when the contingencies become too diverse to anticipate when the grammar is written, that grammar-controlled systems will metamorphose into a more message-controlled style.

The *direct production* school is much closer to the philosophy underlying the present work. Their approach is to start with a data structure from the expert program (their "message") and to evaluate it with a special "text generation" evaluator just as in other circumstances they might evaluate it with, e.g., the normal LISP evaluator in order to execute some function. The structure of the message governs what generation processes are run and in what sequence (invariably a strict depth-first sequence, translating arguments before functions and using the internal LISP stack to record what to do next and what to do with "subtexts" as they are constructed). The "generation functions" for individual kinds of program objects assemble texts by embedding the texts produced for their argument objects within a matrix text; conceptually, generation functions play exactly the same role as dictionary entries in the model presented here. We suggest that the difficulties these systems face—almost complete ignorance of grammar, and an inability to produce text that is not absolutely isomorphic in structure to its message—could be overcome if they were to adopt an intermediate, *linguistically motivated* representation. Such a linguistically motivated representation would, suitably interpreted, serve as a ready description of context and a mechanism for the automatic (i.e. not expressly requested in the message) application of general rules, a policy which, not coincidentally, is the central theme of the present theory.

6.3 When is this linguistic component appropriate?

The utility of an independent linguistic component is that it can be incorporated whole into a new system, relieving its users of the need to develop their own version of this level of the generation process. This utility is not without its price however, since preparing an interface between a new expert program and this component is not a trivial undertaking: creating an adequate dictionary will require the new user to provide explicit representations for relations that are often left implicit in expert programs; and thought must be given to the mechanics of message construction and to the practical exigencies of dealing with another independent computer program. If all that a person wanted to do was to take already highly sugared expressions directly from his or her program's data structures and produce "linearizations" of them, then it is questionable whether they should go to the effort of using this component since the already established "direct-translation" technology should be able to do the job with considerably less overhead.

From an engineering point of view the strength of this component is its ability to combine disparate internal data structures on the basis of their linguistic descriptions to produce cohesive, context-sensitive texts. If, to choose an extreme example, all of the remarks that an expert program were ever going to have to make could be anticipated at the time the program was written, then this component would be entirely superfluous since the texts could be included at the time the program was written. If on the other hand the expert program is continually entering into new discourse situations, learning about new objects and relations, and forced to dynamically configure its remarks to the audience and situation, then using this component (or something like it) is a necessity.

To be specific, a program that needs to produce texts with any of the following characteristics should benefit by using this linguistic component for its generation. These linguistic properties of a text are controlled by a complex set of rules of little interest to the nonlinguist, yet are crutial if the texts are to be natural English. With these rules incorporated once and for all into a shareable component, the system designer is freed to move on to other problems.

Embedded clauses: Any internal relation that is used as part of a description or is modified by or is an argument to another relation, for example the propositional arguments of modal predicates such as "believe" or "possible", will appear as some form of embedded clause when rendered in English. The grammar of these constructions involves complex syntactic rules to coordinate adjustments to the text of the relation and to the matrix text it is embedded in.

Coherence relations in multi-sentence texts: Text that is part of a larger discourse must obey certain linguistic conventions that have no counterparts in the purely conceptual structure of the information being conveyed, e.g. the use of pronouns or definate noun phrases for subsequent references to the same objects, the ellipsis of predictable phrases, segmentation into sentences and paragraphs, the subordination or focusing of individual items, or the deliberate use of explicit relational connectives and ordering to present complex relations sequentially.

Context sensitive realizations: Within a program it is often possible (even desirable) to be vague about whether an expression denotes an object, a relation, or a predicate. The corresponding linguistic choice (e.g. noun phrase, clause, or verb phrase) then depends on how the expression is being used in a given instance, as determined by its context in the message or by the linguistic context into which it is introduced. The choice of realization must be postponed until the context is

clear.

Describing objects from their properties: When a program is continually creating or being told about new objects, pre-stored texts for object descriptions must be abandoned in favor of algorithms that will construct descriptions from properties. For general algorithms, linguistic descriptions of the properties are required to insure that only grammatical phrases are built. Planning is required to judge how thorough a description must be, and the nature of the description selected will effect how it can be realized linguistically. For example deciding between the two texts: *"(I put an X on) the adjacent corner"* versus *"...the corner adjacent to the one you just took"*, the choice between using the prenominal adjective versus using the postnominal adjective phrase depends on the prior choice of how much detail of the position must be given for the audience to recognize it.

6.4 What This Model Can Not Do

Efficiency has its price. Because of its design, there are certain kinds of potentially useful operations that this linguistic component is intrinsically incapable of. This is not taken as a failing, but as the necessary result of a deliberate distribution of tasks according to the components that are architecturally most suited to performing them; that is, I claim (but will not justify here) that the bulk of what this linguistic component cannot do can be done better by other the components that it will interact with. Specifically:

Creative expression—fitting old words to new situations: This linguistic component does not know what words mean. By inverting its dictionary it could compute in what circumstances a word could be used, but it has no means of its own for interpreting these "circumstances" and generalizing them. (How could it if it is able to be used with expert programs with different conceptualizations.) A dictionary entry selects words reflexively according to its precomputed possibilities; in particular, it does not use any sort of pattern-matching on "semantic features", both because of the computational expense and because features that capture useful generalizations are unlikely to be refined enough to pickout specific words.

Monitoring itself: It is generally easier to anticipate and forestall problems by planning than to monitor for them and then have to edit an ongoing procedure. This linguistic component capitalizes on this rule of thumb by omitting from its process architecture the expensive state history that would make editing through backup possible. The kinds of unwanted effects that are difficult to avoid through planning (because they would require essentially full simulation) are coincidental structural or lexical ambiguities; these require a multi-constituent buffer to detect (the sort which is natural to parsers) and are thus better noticed by "listening to oneself" and interrupting the generator with new instructions when needed, rather then burdening that process with a large buffer which will otherwise go unused.

Recognizing when a message will unavoidably lead to awkward or ungrammatical text: Again, given the present design this possibility cannot be foreseen at the linguistic-level without a complete simulation (i.e. rehearsing to oneself). Either the speaker's message-building heuristics will be such that these problems just will not occur (this is almost inevitable when messages are planned and motivated in detail in accordance with the "constraint-proceeds" stipulation), or, by planning the message in terms of rhetorical predicates such as "modifies" or "focus", potentially awkward phrasings will be foreseen at the linguistic level and planned around by general rules.

Reasoning about trade-offs caused by limited expressibility: It can happen that the inability to simultaniously express, e.g., modality and subordination will not become apparent until the

realization of the message is already begun. To be able to reassess the relative importance of the message elements that prompted those choices, this linguistic component would (1) need a common vocabulary with the speaker in which to express the problem (since what should be done is ultimately the speaker's decision), and (2) need to be aware of the potential problem early enough to be able to plan alternatives. Without such a vocabulary, the component must rely on the tacit specification of relative-importance provided in the ordering of the message and the speaker must be prepared for its messages to sometimes not be realized completely.

Planning by backwards chaining from desired linguistic effects: One cannot give a specific grammatical relation as a high-level goal in a message and expect this linguistic component to perform the means-ends analysis required to bring it about; e.g. one cannot give it instructions such as: "the subject of what I say next should be the same as the direct object that I just said". Such reasoning can require exponential time to carry out and a high processing overhead. The effects of such instructions can sometimes be achieved "off-line" however, by having the the designer precompute the decision-space that the deliberation would entail and then incorporate it into the component's library as what would in effect be an extention of the rules of the grammar. (That above instruction, for example, is roughly equivalent to the existing focus heuristic.)

7. References Cited

Brachman, R. J. (1978) A Structural Paradigm for Representing Knowledge. Technical Report 3605, Bolt Beranek & Newman, Cambridge, Massachusetts.

_____, Bobrow, R. J., Cohen, P., Klovstad, J. W., Webber, B. L. and Woods, W. A. (1979) Research in Natural Language Understanding: Annual Report September 1, 1978 to August 31, 1979. Bolt Beranek & Newman, Cambridge, Massachusetts.

Brown, G. (1972) An Experiment in German-to-English Translation. Master's Thesis, Massachusetts Institute of Technology.

Brown, R. H. (1973) Use of Multiple-Body Interupts in Discourse Generation. Bachelor's Dissertation, Massachusetts Institute of Technology.

Chester, D. (1976) "The Translation of Formal Proofs into English." *Artificial Intelligence* 7, 3, pp.261-278.

Clippinger, J. H. (1975) "Speaking with Many Tongues: Some Problems in Modeling Speakers of Actual Discourse." in Nash-Webber eds., Theoretical Issues in Natural Language Processing, Cambridge, Massachusetts.

_____ (1978) **Meaning and Discourse: A computer model of psychoanalytic speech and cognition** The John's Hopkins University Press, Baltimore.

Cohen, P. (1978) A Helpful Computer Conversant. Doctoral Dissertation, University of Toronto.

Davey, A. (1974) Discourse Production. Doctoral Dissertation, Edinburgh University; available from Edinburgh University Press, 1978.

Genesereth, M. R. (1978) Automated Consultation for Complex Computer Systems. Doctoral Dissertation, Harvard University.

Goldman, N. M. (1974) Computer Generation of Natural Language from a Deep Conceptual Base. Doctoral Dissertation, Stanford.

Heidorn, G. E. (1971) Natural Language Inputs to a Simulation Programming System. NPS-55HD71121A, Naval Postgraduate School, Monterey, California.

Halliday, M. A. K. (1966) *Notes on Transitivity and Theme in English. Journal of Linguistics* 2, 37-81.

_____ (1970) *Functional Diversity in Language as Seen from a Consideration of Modality and Mood in English. Foundations of Language* 6, 322-361.

Hawkinson, L. (1975) *The Representation of Concepts in OWL.* in the proceedings of IJCAI-4, 1975, Tibilisi, USSR, pp.107-114.

Kempen, G. (1977) *Building a Psychologically Plausible Sentence Generator.* presented at The Conference of Empirical and Methodological Foundations of Semantic Theories for Natural Language, March 1977, Nijmegen, The Netherlands.

_____ and Hoenkamp, E. (1980) *A Procedural Grammar for Sentence Production.* Technical Report Max-Plank Institute, Nijmegen, The Netherlands.

Labov, W. (1966) *On the Grammaticality of Everyday Speech.* Linguistic Society of America meeting, 1966, New York.

Lehnert, W. (1977) *Human and Computational Question Answering. Cognitive Science* 1, 1.

Mann, W. C. and Moore, J. (1981) *Penman. American Journal of Computational Linguistics.*

_____, Bates, M., Grosz, B. J., McDonald, D. D., McKeown, K. R. and Swartout, I. in press. *The State of the Art in Text Generation. American Journal of Computational Linguistics.*

Marcus, M. (1980) A Theory of Syntactic Recognition for Natural Language. Cambridge, Massachusetts: MIT Press.

Mark, W. (1981) *The Consul Project.* Information Sciences Institute, Marina Del Ray, California.

McDonald, D. D. (1980) *A Linear-time Model of Language Production: Some Psycholinguistic Implications.* abstract in the proceedings of 18th Annual Meeting of the Association for Computational Linguistics, June 1980, University of Pennsylvania.

_____ (1980) *Natural Language Production as a Process of Decision-making Under Constraints.* Doctoral Dissertation, Electrical Engineering and Computer Science, Massachusetts Institute of Technology; Technical report from the MIT Artificial Intelligence Lab in preparation.

_____ (1981) *MUMBLE.* IJCAI-81, August 1981, University of British Columbia.

McKeown, K. R. (1980) *Generating Relevant Explanations: Naturallanguage responses to questions about database structure.* AAAI, August 1980, Stanford University.

Meehan, J. R. (1976) *The Metanovel: Writing Stories by Computer.* Research Report 74, The Yale A.I. Group, Yale University, New Haven, Connecticut.

Minsky, M. (1974) *A Framework for Representing Knowledge.* AIM-306, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Perlmutter, D. M. and Postal, P. M. (in preparation) Relational Grammar.

Roberts, B. and Goldstein, I. P. (1977) *The FRL Manual.* AIM-409, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Schank, R. (1976) Conceptual Information Processing. New York: American Elsevier.

Shapiro, S. C. (1975) *Generation as Parsing from a Network into a Linear String. American Journal of Computational Linguistics,* fiche 35.

Shortliffe, E. H. (1976) Computer Based Medical Consultations: MYCIN. Amsterdam, The Netherlands: Elsevier Noru Holland Inc..

Silverman, H. (1975) *A Digitalis Therapy Advisor.* technical report 143, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Simmons, R. F. and Slochum, J. (1972) *Generating English Discourse From Semantic Networks. Communications of the ACM* 15, 10, 891-905.

Swartout, W. (1977) *A Digitalis Therapy Advisor with Explanations.* Technical Report Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

_____ (1981) *Producing Explanations and Justifications of Expert Consulting Programs.* Technical Report 251, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Winograd, T. (1973) Understanding Natural Language. New York: Academic Press.

Winston, P. H. (1980) *Learning and Reasoning by Analogy: the details.* AIM-520, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

The Yale A.I. Group, (1976) *Annual Research Report,* Yale University, New Haven.