NATURAL LANGUAGE PRODUCTION
AS A PROCESS OF DECISION-MAKING UNDER CONSTRAINTS

by

David Daniel McDonald

S.B. Massachusetts Institute of Technology
(1972)

M.S. Massachusetts Institute of Technology
(1976)

Submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

August 1980

Signature of Author Signature redacted
Department of Electrical Engineering and Computer Science

Certified by Signature redacted
Professor Marvin M. Minsky, Thesis Supervisor

Accepted by Signature redacted
Chairman, Departmental Committee on Graduate Students

Natural Language Production as a Process of Decsion-making Under Constraint

by

David Daniel McDonald

Submitted to the Department of Electrical Engineering and Computer Science on August 8, 1980 in partial fulfillment of the requirements of the Degree of Doctor of Philosophy.

Abstract

A theory of the *linguistic component* of the language production process has been developed, and has been used as the basis of a computer program ("MUMBLE") that produces texts for five experimental "micro-speakers".

The theory is based on a view of production as a goal-directed, decision-making process. The speakers goals ("messages") are interpreted as a program of decisions to be made, ordered according to their relative importance and dependencies. An explicit representation of the results of earlier decisions is used to constrain later ones.

Goals may be described to the linguistic component using any representation the designer chooses provided they can be enumerated as a hierarchy of relations in a fixed vocabulary. A dictionary and set of interface functions must be written to interpret the vocabulary for the linguistic component. Pending decisions and the results of earlier decisions are represented as a *surface-level, syntactic constituent structure.*

Texts are produced by realizing each goal-relation in turn according to the decision-procedure given by its dictionary entry. Decisions specify a sequence of linguistic actions that may be described as the instantiation and traversal of a phrasal schema selected from the grammar. The relation's arguments are mapped into the leaves of the phrase which is then traversed interpreted top-down and left-to-right by a uniform, data-directed controller that produces the text by executing the actions specified by the phrase.

All decisions are local to the linguistic context at the position of the controller. Decisions must be indelible, i.e. not retractable—the controller does not backup. Because of these properties, the process is subject to structurally-induced linguistic constrains that govern when a decision can be made, the kinds of information it may appeal to, and what it may effect. Given certain assumptions about how the speaker structures its input goals, the process can be guarenteed to run at a bounded rate and in linear time with the size of the input.

The use of an explicit linguistic structure as the representation of given and planned decisions, makes it possible to write general purpose rules and heuristics that apply directly in terms of the representational structure and need know nothing about its semantic content. An elaborate grammar has been written including rules for word order and morphology, agreement, subordination, thematic transformations, ellipsis, wh-movement, and subsequent reference.

Thesis Supervisor: Marvin M. Minsky

Title: Donner Professor of Electircal Engineering and Computer Science

PREFACE

This paper presents a theory of how a particular human ability—the goal-directed production of natural language texts—may be formalized for implementation on a computer. Its focus is on issues and alternatives in the design of the theory, and on specific problems in natural language that any theory of production must handle if it is to be adequate. The paper is not a cookbook: the reader will not be given a set of specific instructions for the construction of their own dictionaries and interfaces, nor will the complete grammar or the dictionaries of the completed micro-speakers be included in the text.[1] This is because once one goes to that level of detail the alternatives are no longer forced by the theory but can be fixed according to the stylistic preferences of the individual designer. Instead the paper will use its specific examples as a way to illustrate the generic problems that a designer will encounter, and will describe the alternative solutions that have been tried with their relative merits and trade-offs.

The paper is divided into four large chapters. The first defines the problem: what has been included and what has been left out. It introduces the theory and the relation of the production component to the other components that make up a complete speaker, and sketches a brief example.

The second chapter is a complete exposition of the specifics of the theory. The representational devices and basic procedures are developed "bottom-up"—begining with the definitions of the data-types for linguistic objects and the dictionary, then describing the control structure and facilities for attached procedures, followed by the principles of the grammar. The exposition is exaustive, and the reader may wish to skim it. Reading the first paragraph of each subsection should provide enough of an overview to make it possible to read the later chapters without having to follow all of the detail.

Chapter three looks at specific phenomena in the grammar of English. The emphasis is not so much on the analyses themselves as on how an analysis must be interpreted in order to be incorporated into this theory of production. The dependency of alternative analyses on the ability of the speaker to supply messages with a particular structure is explored.

Chapter four looks at the design of the dictionary. A taxonomy of message elements is developed in terms of how they are interfaced to the linguistic component. A range of representational devices for the dictionary is given with examples of their use, and with regard to the possibilites for sharing parts of the dictionary, ultimately leading to development of an

1. The programs for the grammar, dictionaries, and control are presently being revised to bring them into line with the refinements that arose while writing this thesis. People who are interested in the actual code should write to me after the beginning of 1981.

interlingua of commonly understood relations. Specific experiences with the micro-speakers are discussed in an appendix.

Subjects covered elsewhere

The literature in language production by machine is reviewed in a separate paper, [ddm_past]. That review covers the period up to August of 1977, and is presently being updated. Some discussion of how the computational theory described here might be used as the basis of a psycholinguistic model of human language production can be found in [model], with some initial observations appearing in [acl80]. This is for me an area of active research, working with evidence from naturally occuring errors in speech and some inferences about on-line processing drawn from hesitations and self-editing. However, the research is preliminary and will not be reported on in this thesis.

Table of Contents

CHAPTER ONE

Introduction

1. 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,[1] we are accurate,[2] and we are unaware of the mechanics of how we do it. By studying language production as a computational problem, as by studying any other natural ability with these properties such as vision or manipulation, we can gain insights into the workings of the mind.

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 (phones) 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 somehow to attend simultaniously 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.

---

1. The Guiness Book of Records speed record for reading English is 400 words per minute. A normal speaking rate is in excess of 160 words per minute or 8 to 10 sylables per second.
2. A study by Labov [labov_everyday_grammaticality] 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.

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—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 for the weakest representational devices that can do the work: devices from whose computational properties the characteristics of human language production will inexorably follow. By doing this, by restricting the kinds of behavior that our model is capable of, we can extract predictions from it and make it subject to empirical tests.

The theory that I will propose in this thesis revolves around the specification of a *processor*, an abstract device that controls every aspect of the timing, the scope, and the access to information of all actions that occur during the linguistic part of the production process. This processor has specific limits to its abilities, limitations that follow directly from the way in which it represents its computational state (i.e. its plans, its history, its current context) and its state-transition algorithm. We can describe these limitations succinctly with the following hypothesis:

> **Language production is performed by an *indelible*[3] process that produces texts incrementally and in their natural order. As a consequence of these constraints as embodied in the control structure of the proposed theory, production takes place at a bounded rate and in time proportional to the size of the message to be conveyed.**

In specifying how the process is controlled, the theory does not attempt to delimit either the facts of English grammar that process obeys or the specific usage heuristics that dictate what will be said in given situation. They are too large a problem for one researcher or even one generation of researchers to presume to solve. Instead, the processor is designed as an interpreter: a constant "kernal" process that appeals to an outside grammar and body of heuristics for the specifics of what it will do. The theory dictates how the grammar and usage-heuristics are to be represented and thereby how they interact within the process; Fixing the representation they can use has the effect of indirectly specifing the facts themselves, since not every fact that one might imagine can actually be expressed.

1.1 Language production as decision-making — two alternatives

As part of the model, we must characterize what kind of a computational problem we take language production to be. Is it a search process that explores a space of possible output texts? Is it a recursive evaluation process that operates without a global context? Different characterizations have lead to very different process architectures and very different representations for the same facts. As should be clear from the title of this thesis, I am modeling

3. In an indelible process, no working structures may ever be changed once they have been made—they have been written with "indelible ink". The theory of natural language parsing developed by Mitch Marcus [Marcus_book] incorporates this same notion of indelibility as an integral part.

production as a decision-making process. The properties of decision-making then will dictate what the theory must be concerned with.

Making a decision requires information, both generic and particular, which can vary in its accessibility and its form according to the state of the process; A decision may be the joint product of several information sources, each of whose contributions may be differently affected by context; The outcome of a decision must have a representation, one that prominantly indicates what further actions and decisions are to be taken; Individual decisions may be dependent of the outcomes of other decisions and may not be able to function or may have very different outcomes if made before those other decisions.

As a consequence of these properties of decision-making, the theory will be very concerned about modularities in the distribution of the needed information; representational levels will be dictated by what kinds of information are needed simultaniously. The usual linguistic representational devices—constituent structure, syntactic categories, grammatical relations, transformations, morphological rules—will be reinterpreted as constraints on decisions. The design of the control structure that dictates when the individual decisions will be made will have enormous theoretical significance since it will be what determines whether decisional-dependencies are followed or worked against.

As we shall see, the indelibility hypothesis is a strong forcing function on the control structure of a decision-making process: under it decisions may not ever be retracted (though they may be refined and embellished). Consequently if decisions are to be made optimally, the control structure must see to it that no individual decision is made before all of the other decisions on which it depends are complete. This fact has the important corollary that many of the standard computational decision-procedures are unsuitable for use in language production, at least as they are ordinarily formulated; these include: relaxation,[4] hill-climbing, hypothesis and test with backup, and backwards chaining from goals. Without these techniques it is not obvious, *a priori*, that goal-directed language production is possible under such a hypothesis. The theory and program of this paper are offered as evidence of its sufficiency.

4. Any parallel algorithm for decision-making must cope with the fact that the dependencies between decisions in production—constraints imposed by earlier parts of the discourse and by the more important goals of each message—impose a very nearly sequential ordering on the decisions if they are to be indelible. A parallel progressive refinement technique such as Mark Stefik [stefik] has developed that uses the posting of constraints between many simultanious active decision-makers to narrow down its choice set could possibly be used in an indelible language production model provided that some natural way could be found to insure that texts appeared in their natural order rather than just according to which decision completed first. The fact that people do not appear to employ progressive refinement at least over large large bodies of text (i.e. multiple clauses, see page <planning_versus_editing>) is probably related more to the kinds of intermediate representations actually available to people than to theoretical difficulties with its control structure.

1.2 What kind of production to model

The character and time-course of human production varies tremendously according to the circumstances, e.g. from writing a difficult essay to fuzzy conversations over breakfast. This variation involves speed, the possibilities for editing, the availability of external memory, and the speaker's conscious involvement in the process. Rather than attempt to capture the entire spectrum of language production in a single theory, I have singled out what I take to be the "basic" production process, and assume that the variations come about through interactions between the basic process and other processes such as the speaker's language comprehension process ("listening to oneself") or a planning process with a source of external memory (e.g. pencil and paper). It is suggestive to heuristically identify the basic process with the subjective "immediate mode" described below. Bear in mind however, that introspection and existing psycholinguistic data do not even begin to determine the details of any effective processing model. The identification is therefore only intended to focus the readers expectations when assessing this theory.

In everyday conversation, we operate in *immediate mode*—we speak without rehearsing what we will say and with only a rough sketch of what we will say next. Similarly, when writing a paper or reflecting on what to say, it is a common experience for phrases or even multi-sentence texts to "spring to mind" in their final form as though we were actually hearing them, and without our having made any conscious effort to form them from their constituent parts. If, on reflection, we decide a phrase will not do, we do not consciously break it down and fashion the changes, rather, a new phrase appears in our mind with something like the modifications we want.

The identification of immediate mode as the guideline for determining the proper level of capabilities for the "basic" process is a post-hoc judgment. My original work on the computer program was not intended to match any particular human behavior but just to provide for the necessary decision-making within a suitable environment. It developed that the features of the program's design which made it favorable for decision-making lead to *inherent limitations* on the amount and kinds of information that were available from moment to moment. These limitations parallel those of people in immediate mode, e.g. words can not be taken back; there is only a limited lookahead;[5] and the program, like a person, sometimes talks itself into a corner. This coincidence of abilities, as coarse as it is, is intuitive evidence that this synthetically derived theory is plausible as a theory of the human production process as well as for language production in the abstract.

---

5. In referring to lookahead limitations in people, I am considering phenomena like these: while writing the first draft of a paper you reach the end of a complex sentence and realize that the sentence would work better if it began with the phrase that you just ended it with; or, later in the paper after you make a particular point, you realize that for that point to be properly understood, you must go back to a much earlier paragraph and expand some definition. With unlimited lookahead capabilities, these problems would have been foreseen and handled correctly the first time.

2. The Linguistic Component

At the basis of this research is the hypothesis that the linguistic part of production can be legitimately and profitably established as a separate component within the process. For this separation to be sensible, it must be the case that the interaction between the "linguistic component" and the rest of the process can be precisely specified and that the amount of "cross-talk" between them—the degree to which they must share assumptions about representations and contingencies—is small.

The other components of the process—the expert program, the speaker, and the language understanding component—are not developed in this thesis except to the extent required to support the "micro-speakers" (next section) used to test the linguistic component.

The expert program From a practical point of view, the driving force behind the development of a sophisticated language production capability has been the emergence of interactive, knowledge-based consultant systems. Programs like MYCIN [mycin], DIG [digitalis], and MACSYMA [macsyma_ref], to name just a few, must communicate fluently with user-specialists who do not necessarily have the time or the motivation to learn programming languages or a special keyword syntax. Consequently, as the mountain to Mohamed, it is the program that must learn a natural languages rather than the casual human user a computer language. The computer program I have written to demonstrate and experiment with the model In psychological terms, we can think of the expert program as the locus of all of the non-linguistic motives, concepts, representations, and decision-procedures that ever come into play in production.

The speaker The primary function of these expert programs is to solve problems in their own domains—not to speak English. The data structures they use and the procedures they follow will have been designed for the convenience of solving their particular problems and will not, by and large, already incorporate procedures for formulating well chosen remarks to the user at appropriate points. Instead, I will assume that such procedures are incorporated into a separate component which I will call *the speaker.* (This is a conceptual distinction; whether or not the speaker and the expert program are physically separate programs is not relevant to any of this work.)

For the most part, a speaker component will be specialized to the domain and conversational situation of a single expert program. Its job will to know when something should be said and to plan the content of the utterance(s) (a representation of this plan is what will be sent to the linguistic component as the "message"). It will decide what propositional content and speech act(s) to express, which details must be included in the utterance and which can be assumed as common knowledge, and whether any other constraints such as ordering, highlighting,

or even word choice, must be imposed in order that the utterance will have its intended effect on its particular audience. (See [cohen_thesis] for discussion of how this planning could be done.)

The language understanding component Presumably, any program that can speak to its users in English can also understand it, and will include facilities for parsing and semantic interpretation. However, I will have nothing to say about such facilities in this thesis. Conventional wisdom says that the two are closely related and it has been suggested that they should use the same grammar [kay_shared_grammar], but no practical proposals have been put forward. My own inclination is to believe that a process-neutral encoding of the relation between the possible natural language constructions and the speaker's intentions—what the constructions can be used for—could be profitably shared,[6] however, the translation of a grammar into actions will necessarily be different for production and comprehension if we are to take advantage of efficient algorithms.

In other respects, there appears to be a growing convergence of the theoretical claims that are being made about the computational power of the language processor. Like this generator, the parser designed by Mitch Marcus [mitch_thesis_book] is indelible—never retracting a decision once it has been made, and both theories make critical use of David Marr's "Principle of Least Commitment" [marr_least_commit] to control how much information they try to decide at each point. The fact that such highly constrained processors can be made to work is support for hypotheses that the structure of natural language is sensitive to processing constraints.

2.1 Between the speaker and the audience

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.

6. In the present design, this is the information that is represented in the dictionary.

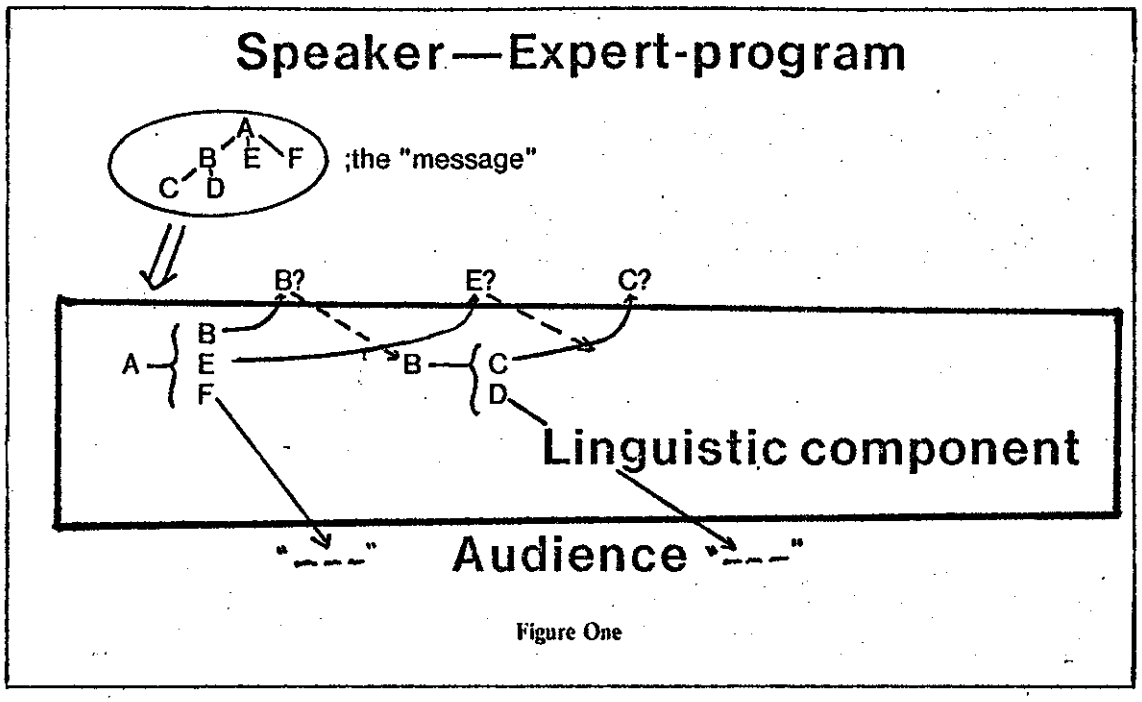


Figure One

The linguistic component understands the messages it receives as programs that dictate what decisions must be made in order to produce an English[7] text[8] that expresses what the speaker has intended. As in any program, the hierarchical structure of the elements in the message is critical. The initial, more important or more encompassing elements will be realized first, creating an environment that constraints, grammatically and rhetorically, the options available for later elements. In the sketched message in figure one, element A is most important (perhaps it identifies special roles that B, E, and F are to play) and it will be realized first. The message is decomposed one level at a time, with lower levels remaining unseen until they are reached; thus A is realized in terms of some linguistic expression involving B, E, and F, but without any awareness at all of B's "subelements": C and D.

While it is decomposing the message, the linguistic component is not completely cut off from the speaker. As diagramed in the figure, the component may at any point ask the speaker questions about a specific message element in order to determine facts that are important only to it (for example "person" and "number") or to further decompose some element, in effect extending the message. The speaker is always accessible but the timing and the computational

7. This research has only looked at one natural language, English. In the future, it will be very important to see how this theory of production is affected by facts from other languages, particularly SOV languages such as Japanese or Dutch. Unfortunately however, building a production facility that is fluent and appropriate requires designers of native fluency in the language used, and I am only fluent in English.

8. For practical reasons, this research has involved only the production of texts. The further problems of producing actual actual acoustic signals with appropriate stress and intonation have been considered but the work is too premature to be discussed here.

context in which the speaker is 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 it 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, perhaps in reaction to what it hears while "listening" to the component's output or monitoring it in some other way, is a question I will leave open. There are some eventualities (such as structural ambiguities) that are difficult to foresee when developing a plan for execution by a linguistic component of this design, and there are also possible divisions of effort within the speaker's planning process which might benefit from a "feedback" design of this sort.[9] 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 concerns of my research.

The output of the linguistic component—the English text—is produced in a continual stream during the whole time that the message is being realized. Left-to-right order is reflected in the order of subsequent decisions and refinements from the first moment that it appears in the linguistic plan, since already spoken text is an important source of constraints on later decisions.

2.2 A cascade of two transducers

As an automata, the linguistic component is best described in terms of two transducers, with the output of the first becoming the input of the second. The first transducer goes from the message to a surface structure level 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.

9. A very interesting model of the production of psychoanalytic developed by Clippinger and Brown [clippinger_erma_tinlap][clippinger_erma_bood] [brown_erma] 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.

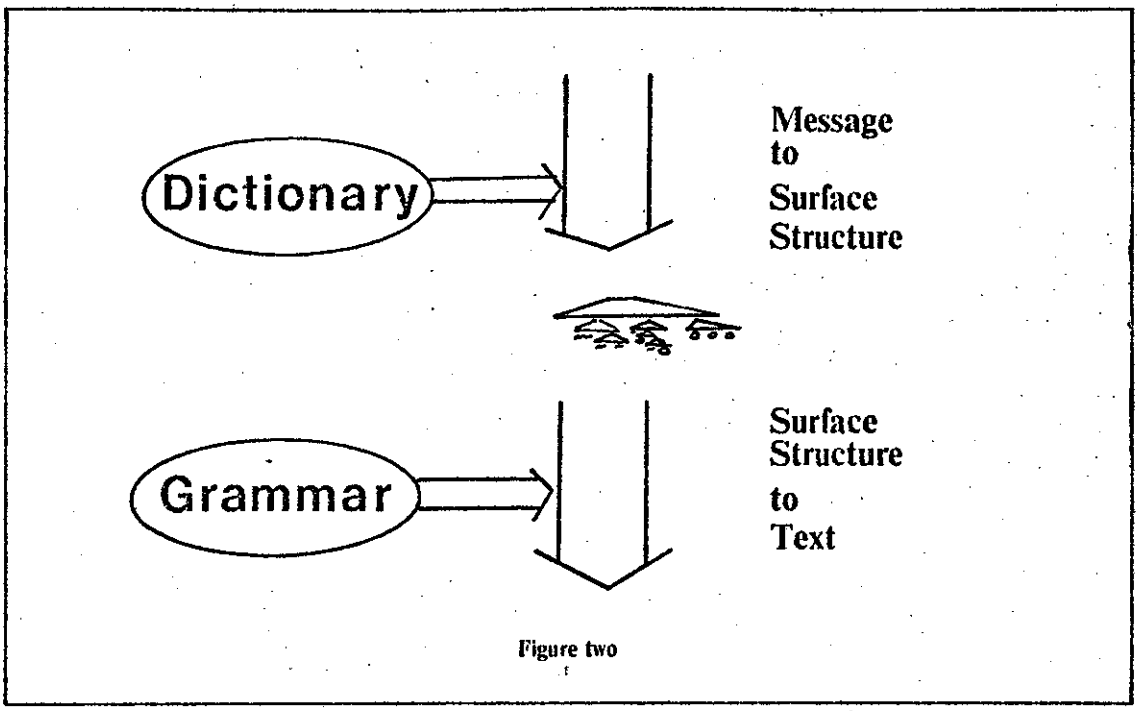


Figure two

Both the message and the surface structure are treated as totally ordered sequential streams of data; only a single token is processed at a time, and it is processed only once. The two streams are processed "on-line", the output from the first transducer for one token being completely consumed by the second transducer beore the first moves on to its next token. The individual transducers are simple finite state machines—they have the ability to traverse their input streams and bind contextual variables but little else. Their transducing powers come 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 procedures and schemata in these two "libraries" are what do all of the real work of the linguistic component; the transducers are responsible for the controlling when the libraries are used and for maintaining the linguistic context to which they will refer. Since, as we will see, the definition of the context and the order in which actions are taken are the key to the linguistic component's representation, the structure of these transducers will be all important to the theory.

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 message 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.

Two transducers, one controller The special property of this cascade is that the two transducers have been folded into a single process: the traversal of the surface structure. The message starts out as the sole constituent of the root node of the surface structure tree; it is *replaced* in that position by its realization, with the realizing phrase incorporating at its fringe the next level of message subelements. A single controller traverses the surface structure in the normal top-down, left-to-right order, dispatching to the grammar for the execution of its linguistic content and to the dictionary for the realization of the embedded message elements. 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 stipulation on the structure of messages (the "well-formed-ness condition", page <well_formed_ness_condition>) which dictates that the enumeration order of a message (the order in which its elements will be realized—a reflection of its hierarchical structure) must be such that any message element that will make reference to other elements in its realization must in fact be realized before any of those elements—there can be no loops in the enumeration. With the enumeration of a message guarenteed to be a tree, we can embed its realization within another tree and be assured that its traversal will be continuous.

2.3 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.* Once we have seen its notation and understood its relationship to the grammar and the realization process (i.e. the dictionary), we will move on to a sketch of the controller, showing exactly how it traverses the tree and how the tree is used to indicate the proper routines to dispatch to in the grammar and dictionary. When that is done, we will look closely at how the tree and the controller are used as the linguistic component processes a simple example message.

Figure three 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 page <snapshot_after_trivial_msg>.) Two kinds of structures are indicated: *constituent structure* to define how the pieces are connected and how the controller is to traverse them, and and *grammatical labels* to describe the linguistic properties it is intended to have.

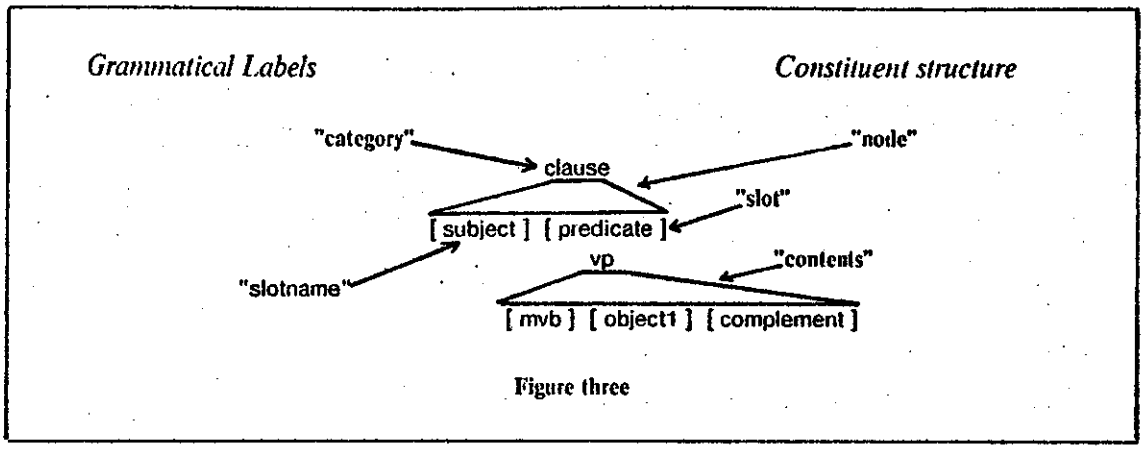


Figure three

Constituent structure is indicated graphically by the pattern of trapazoids and brackets: the trapazoids indicate "nodes", and the brackets positions of possible constituents referred to as "slots" or "constituent slots". The actual constituents themselves are the slots' "contents". The node labeled "vp" is the "predicate constituent" (also abbreviated "[predicate]") of the "clause node". Besides being a node, the contents of a slot may be a word, or a message element, or they may be empty. A node consists of its categories and the slots for its "immediate constituents" (i.e. just those at the bottom of the trapazoid). A subtree from a given node to the fringe of the tree will be referred to as a *phrase.*

Grammatical labels will either label nodes, in which case they will be called "categories" and printed just above the trapizoid; or they will label constituent slots, in which case they will be called "slotnames" and printed inside the brackets. A node or slot may have more than one label. This constituent structure representation is different than 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 is also done in so-called "relational grammar" [relational_grammar] and was used in some early phrase structure systems [postal_phrase_str_review]). The "subject" constituent, for example, would be defined as the noun phrase node directly under a clause node. Slotnames are very important however because they are what carry the grammatical properties of the constituent positions; a comparably powerful relative position scheme would either an unbounded computation as the tree grew in depth or an undue multiplication of category names—the use of slotnames is a more natural treatement of grammatical functions.

The consituent 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.

The importance of this path is not in the nodes and slots it traverses but 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 it 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.

Growing the tree

The tree is a temporary structure created by the realization process (the first transducer) and garbage collected once it has been used. Figure five shows the initial configuration of the tree just as a message (the trivial one we will use as the example) is being added to it to start the linguistic component processing.

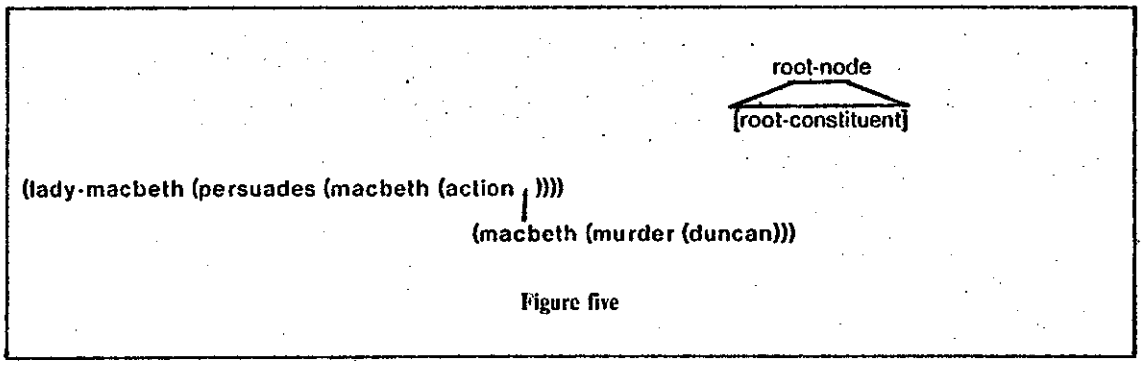


Figure five

The "root-node" and its one "root-constituent" are the only permanent parts of the tree. They have no grammatical properties and are just a way to "tie off" the tree in a consistent manner.

A preview of the realization process The tree is extended whenever the controller reaches a message element at the fringe of the tree (actually an "instance" of a message element, see section <elmt_instance>) and the element is replaced as the contents of its slot by a newly created node.

This node appears as the last step in the realization process, which begins (if this is the first time the message element has appeared) with the interpretation of the element's *dictionary entry*. (The realization process may also result in a single word, or in another message element.)

A dictionary entry consists of a set of possible "choices" and a set of "decision-rules" to pick between them, where a *choices* is a symbolic specification of phrases, words, or subelements of the element being realized, and a *decision-rule* is a list of predicates that may examine both the linguistic context and the context of the speaker and 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 (see pg.<transformations_intro>).

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 the legitimate sequences of slotnames that it can have. These listings are organized in terms of "constituent schemas". Every choice has (at least) three parts: (1) a "phrase-schema" that defines a tree of constituent-schemas possibly augmented by additional slotnames or categories (often referred to as "category-features" and "slot-features", or just "features") 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 six lists all of the structures from the grammar and the dictionary that would go into the realization of the example message element.

To produce new constituent structure from the choice, its phrase-schema must be instantiated and the mapping applied to it. The result for this example (now knit into the tree) is shown in figure seven.

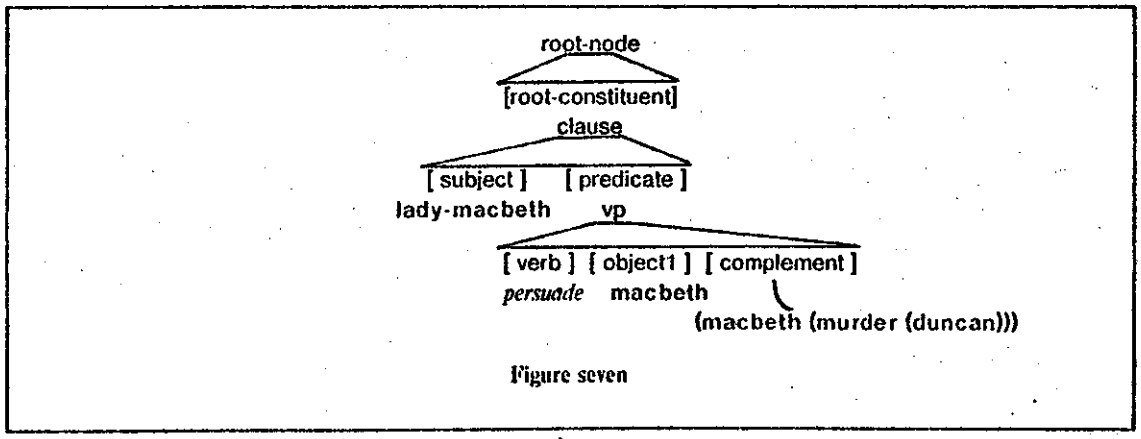


Figure seven

Message Element

```
(lady-macbeth (persuades (macbeth (action ))))
                                        (macbeth (murder (duncan)))
```

Dictionary Entry

```
(define-entry persuade-entry (frame)
     variables ((persuader (car frame))
                (persuadee (cadadr frame))
                (action (cdadr (cadadr frame))) )
     default
       (clause-DO-COMP_persuade persuader
                                persuadee
                                action))
```

the Choice

```
(define-choice clause-DO-COMP_persuade
     parameters (subj do comp)
     phrase (basic-clause ()
                  predicate (vp-do-comp ()
                                verb "persuade" ))
     map ((subj . (subject))
          (do . (predicate object1))
          (comp . (predicate complement))) )
```

Constituent-schemas

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

(define-schema vp-do-comp
     categories (clause)
     slots (verb object1 complement))
```

Figure six

2.4 The Controller

The controller has two functions: (1) to execute the "plans" defined by the input data to the two transducers, and (2) to maintain the computational environment expected by the procedures in the dictionary and grammar. Only the knowledge base for the first function needs to be part of the actual controller, as the maintance of the environment will really be done by the grammar-routines with the controller acting only as a repository. Even the plan execution is very simple, since all that the controller must know is how to traverse the tree and how to access the grammar and dictionary based on what it reads in the tree.

The tree-traversal algorithm is given by the flowchart in figure eight.

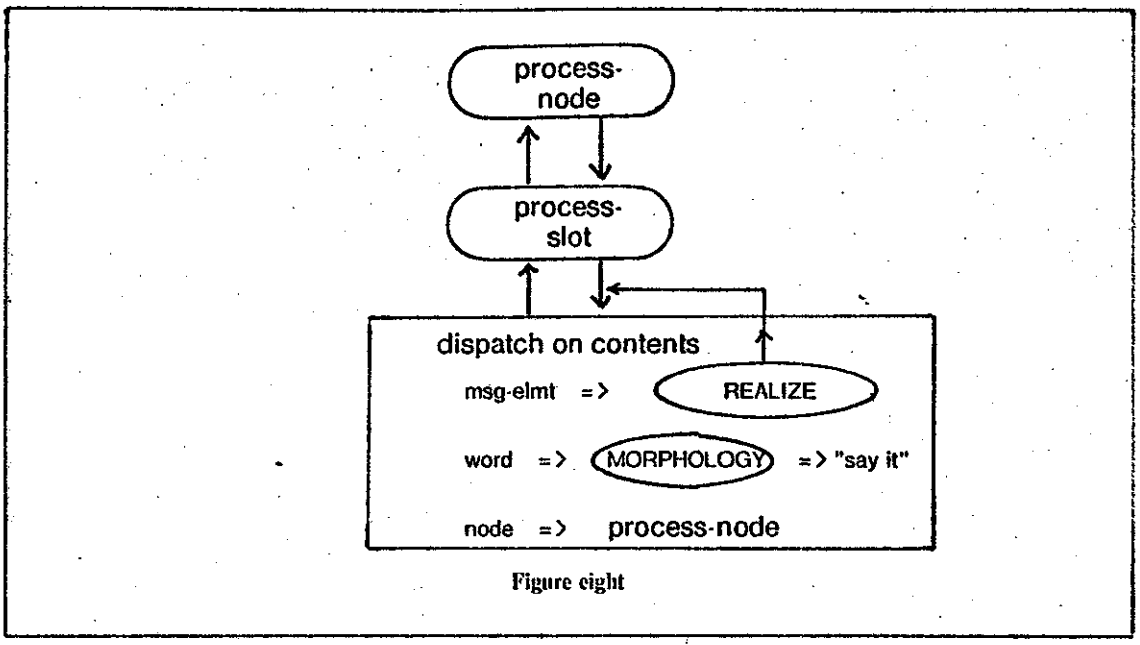


Figure eight

Moving from one step to the next in this algorithm is a matter of reading-off the appropriate properties from the nodes and slots in order to determine where next to go. (The complete flowchart for the controller is given in chapter two on page .)

The first transducer is taken up exclusively in the dispatch to the function "realize". Every message element that is embedded in the tree must eventually pass through this step of the controller, at which point it will be passed to the realization process via that function. After the message element's realization—i.e. the node, word, or other message element returned by realize—is knit into the tree, the controller loops around and repeats the dispatch on the new contents of the slot.

The second transducer is a bit more widely distributed. Every grammar-routine is associated with two items: (1) a label from the set of constituent structure labels, and (2) an "event" within the controller. There are five generic events in controller; they are defined in terms of controller's passage through the constituent structure and correspond, naturally enough, to the different kinds of constituent structure labels.

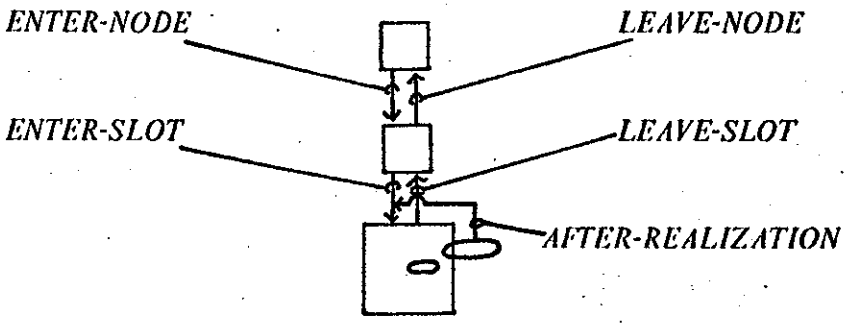


Figure nine

The attachment of the events to the arrows between the boxes is really a pedagogical exageration; the boxes are really constituted of nothing more than the execution of the routines associated with these events. X-node and X-slot events are associated only with categories and slotnames respectively; after-realization events are associated with slotnames.

The current context The rules of grammar embedded within the grammar routines will be 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 by the tree but have their values set and reset by of specific grammar-routines. The first three variables are referred to as *controller-variables*, and the second, open-ended set as *grammar-variables*. In addition to the variables, the controller maintains a *discourse history*, consisting of records of all important events that have occured, including the realization of every message element instance, every selected choice, and every decision brought about by the grammar (pg.<grammar_decisions>).

In summary, the current context of the lingustic component can be viewed as a four dimentional 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.

2.5 From message to text — an example

At this point, we have accumulated enough machinery and terminology to follow an example message through from beginning to end. The message I will use is the one we saw earlier about Macbeth and Lady Macbeth. The transparency of its contents will be an advantage since it means that we will not have to dwell on domain-specific decision procedures and can concentrate on the linguistic processing. The message comes from the "Macbeth Domain" described in the

next section of the introduction. It was originally part of a frame describing the properties of Lady Macbeth as a character in the play. Taken out of context like this, the property does not draw any thematic transformations and thus will be realized as a text with almost the identical "content" as its message:

"Lady Macbeth persuades Macbeth to murder Duncan."

Roughly all that has happened is that we now have mixed case English words where we formerly had hypenated LISP atoms; the verb "*persuade*" appears in the present tense singular agreeing with "*Lady Macbeth*"; there is punctuation; and the embedded property, "*Macbeth murders Duncan*", has been subordinated.

With this simple example, we can proceed via very close snapshots and still list practically all aspects of the current context each time. We will begin the example at the point just after the original message has been realized (as described above) and before the controller has begun to traverse it.

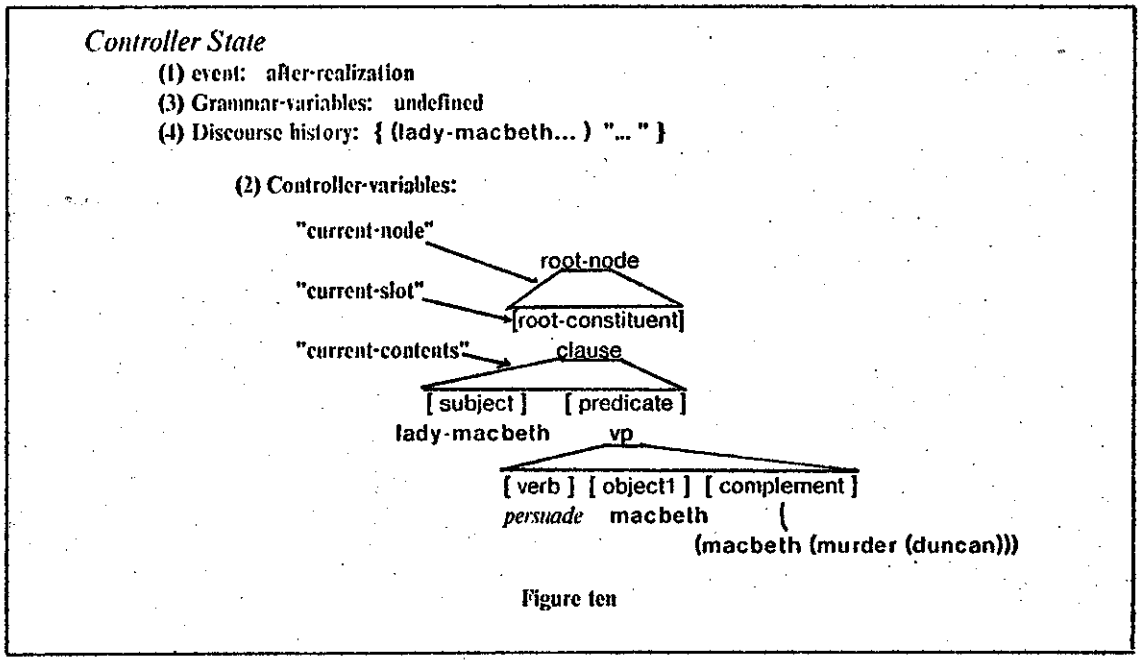


Figure ten

Only one event has happened so far, the realization of the message as a single clause phrase embedding three subelements. This event is recorded in the discourse history but I will not confuse the reader by including the details of the record; it is discussed in section <discourse_history>.

With the controller in this state, the next thing that happens is the dispatch on the "type" of the value of the variable current-contents. As this type is "node", the controller calls its process-node routine recursively and we proceed to look for any grammar-routines that are indexed by the

event "enter-node" and the category "clause". There are several of these, and their effect is to set various grammar-variables. The next snapshot shows the state of the controller after we have passed through the clause node and subject slot and are about to send the current-contents—lady-macbeth—to the realization routine.

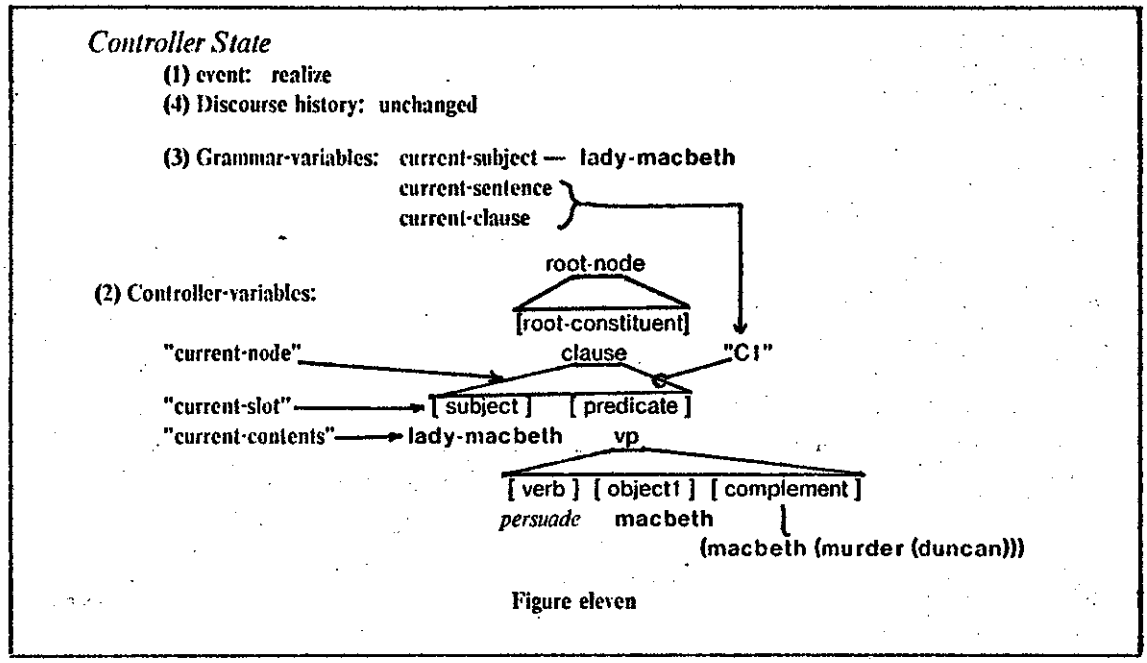


Figure eleven

Two fine points: the value of the "current-subject" will always be the message element that originally filled the slot rather than the noun phrase that replaces it. This is because the grammatical properties of the subject with which we are most concerned, here the influence of the subject on the embedded property, revolve around the subject's identity at the message level; by making our record of "current-subject" when that level is readily available to us, we have a simpler and more efficient process. The second point is that the two other grammar-variables are pointing not to clause node itself, but to a record of it. The nodes and slots of constituent structure are strictly temporary objects that may be garbage collected as soon as the controller has passed through them, while the records of the nodes include only that information that the grammar writer believes will be useful grammatically and will be retained until they are no longer referenced by any of the parts of the controller.

The message element lady-macbeth is realized trivially by an entry comparable to the one shown for the whole property. It has no decision-rules only a default choice, namely to use the

word "*Lady Macbeth*".[10] A dictionary entry this simple can and should be schematized to ease the job of the dictionary designer; the designer writes down the first expression below and when the system is loaded it is expanded into the second expression with the customary load-time syntax for entrys.

```
(def-word-entry  lady-macbeth   |Lady Macbeth|)

(define-entry  lady-macbeth_entry  ()
     (matrix
          default  (use-word  "Lady Macbeth"))))
```

Once this word is returned and knit into the tree, the controller immediately loops around and passes it to the morphology routine to be printed out.

The next snapshot catches the linguistic component several constituents further along, after it has left the [subject], entered the [predicate], and passed through the first two of its constituents.

10. "*Lady Macbeth*" appears in the vocabulary as a single word even though it is printed out as two. Because it is "atomic", the linguistic component is unable to notice that that the lastname "*Macbeth*" will appear twice in this text, and thus will not automatically consider using an alternate phrasing such as "*her husband*" as a stylistic variation even though a speaker like the Macbeth domain would be able to supply the necessary supporting information. Rules of this sort can be expressed in the linguistic component through the use of well-annotated dictionary entries and the shared "subsequent reference" routine within the realization process. See section IV.C.3.1 for further discussion.

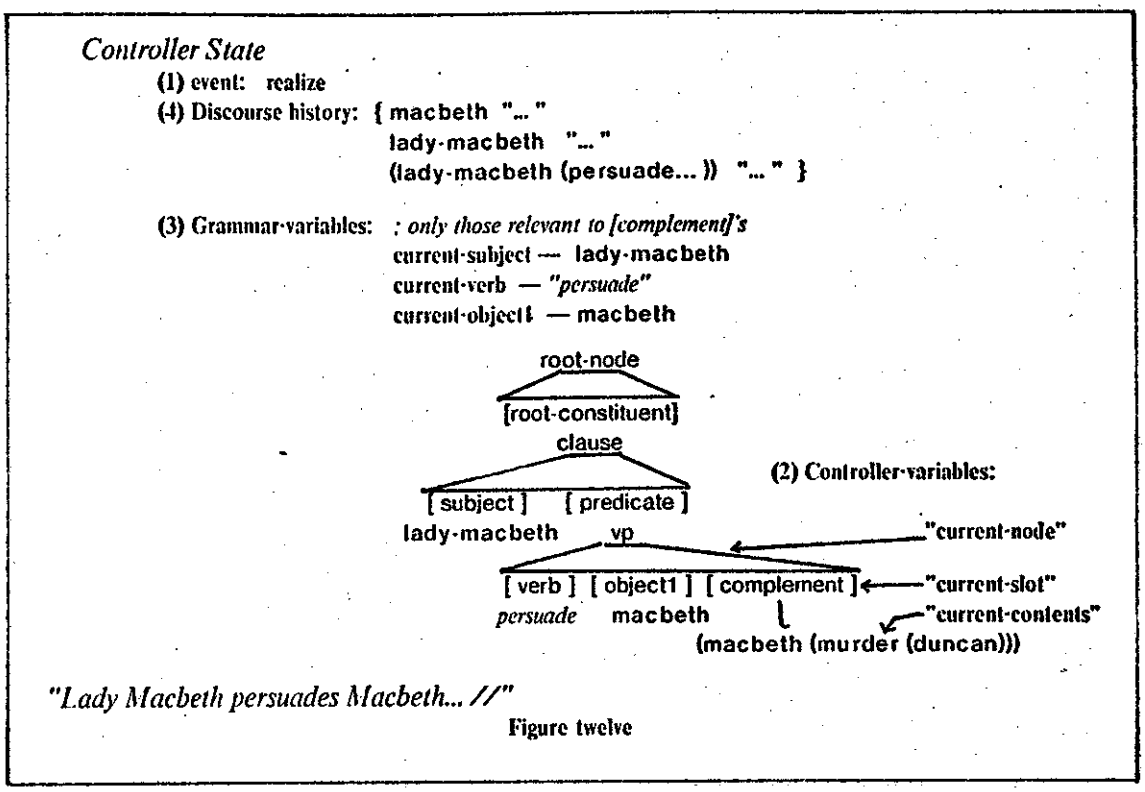


Figure twelve

The dictionary entry for a murder consists only of a default choice, just like the earlier entrys. The difference is that clauses in a complement slot must undergo some kind of "embedding transformation" if they are to be grammatical. The need for the transformation has nothing to do with the properties of the message element leading to the clause (though the particular choice of transformation may be sensitive to them); consequently, we should have the transformation process apply transparently to the entry-based realization process. This is done by introduction tests for transformations and their application as a regular step in the realization procedure, as we can see from the flowchart in figure nine.

The first thing that occurs in the realization procedure is a test to determine whether or not this is the first instance of the message element to appear in the tree, which it is. Had it been a subsequent reference, we would have taken the other fork of the procedure and tested whether we could have used a pronoun. Within the "main stream" we interpret the element's dictionary entry in order to arrive at the realization. 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. The entrys of this example, of course, have only matrix decisions.

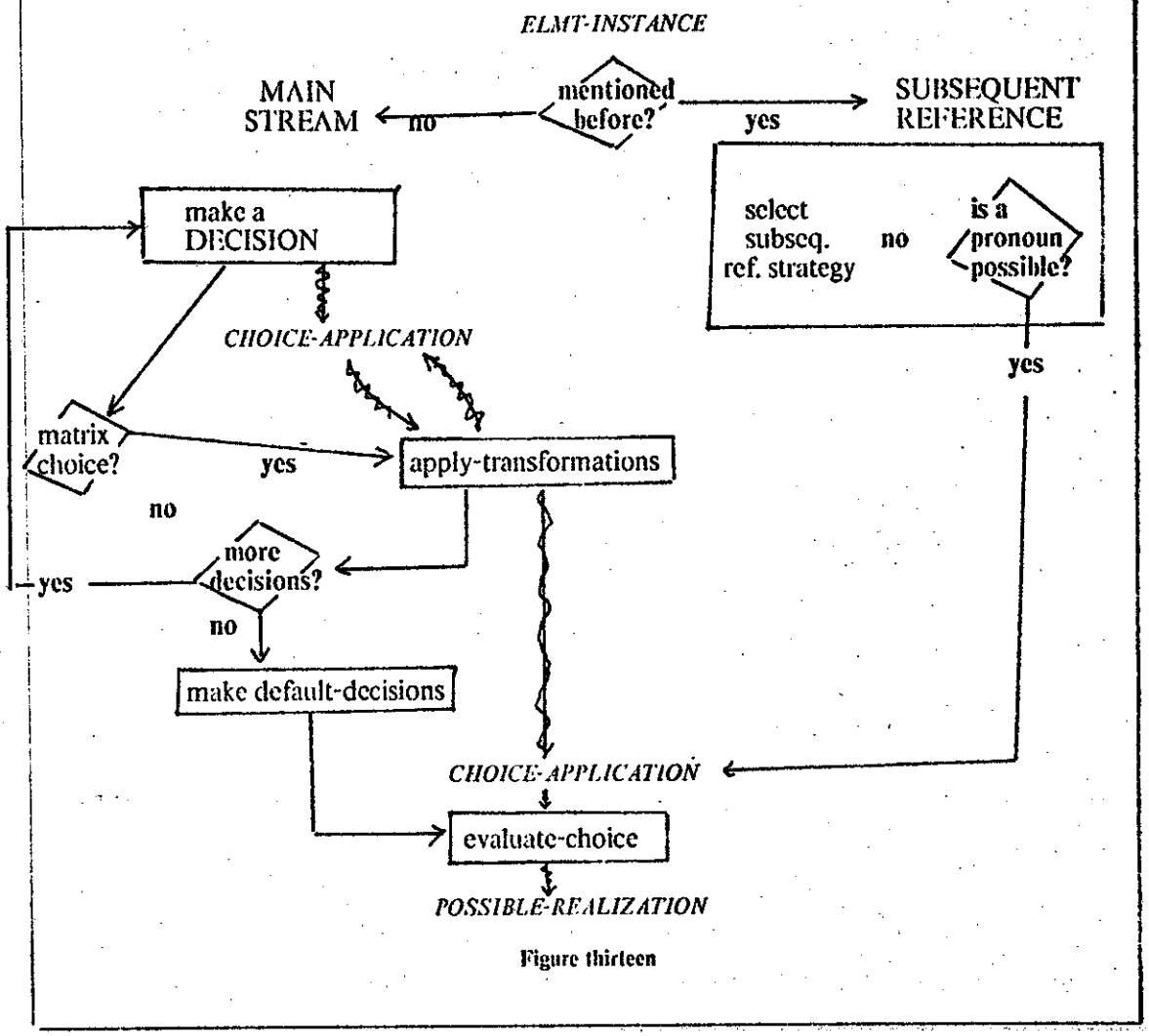


Figure thirteen

Once the matrix decision has been selected, here by default, the "entry-interpreter" looks to see what transformations, if any, are associated with the choice. This association is on the basis of the pattern of slotnames specified in the phrase-schema as shown in figure fourteen. Because the phrase-schema includes an object slot, it may be subject to the passive transformation; because it includes a predicte slot, it may be subject to verb phrase deletion; because it includes a subject slot, it may be subject to equivalent noun phrase deletion; and so on. Just as with the regular decisions of an entry, each of these potentially applicable transformations is associated in the grammar with a set of decision-rules. At the time the grammar and dictionary are initialized, a postprocessor is run over the entrys and their choices in order to compile one monolithic decision procedure for each combination of potential transformations that ever applies to a choice. Within this procedure the order in which the different transformations are considered is fixed and provisions are made to allow new transformations to be included after the application of a

transformation changes the constituent structure labels of the choice so as to make those transformations possible (i.e. more than one transformation can be applied to the choice).

In this case, passive will not apply because the element is not appearing in a thematically marked context (section III.1.2), and verb phrase deletion will not apply because it is not within a conjunction or other "coordinated slot" (section <ellipsis>). Equivalent noun phrase deletion could apply, however, because the current-slot is [complement] and complements take only subordinated clauses as constituents. Because the subordinating slot is [complement] and the grammar of complements is distributed between the slotname and the complement-taking verbs (section <encoding_facts_into_slot_names_versus_finding_them_in_the_context>), the decision procedure now looks to the properties of the current-verb to determine what the range of legitimate subordinating transformations actually is.

Complements to the verb "*persuade*", may either be "that-complements": "*I persuaded myself that I should go home and go to bed*" or "infinitive complements": "*I persuaded myself to go home and go to bed*". The choice between them is presently analyzed as a "marked"-"unmarked" distinction: the that-complement is selected if the message element being realized is explicitly marked to express tense or modality, otherwise the infinitive complement is selected, as it is in this example.

APPLYING TRANSFORMATIONS

(1) Make the "matrix" decision of the entry

default (clause-OBJECT1_murder...)

(2) Is the phrase-schema of the choice associated with any a transformations ?

[subject] -> "equivalent-np-deletion"
[predicate] -> "verb phrase deletion"
[verb] -> "gapping"
[object1] -> "passive"

(3a) Evaluate the combined decision-procedure for the transformations

{... } -> "equivalent-np-deletion"

(3b) and apply any transformations whose conditions are satisfied.

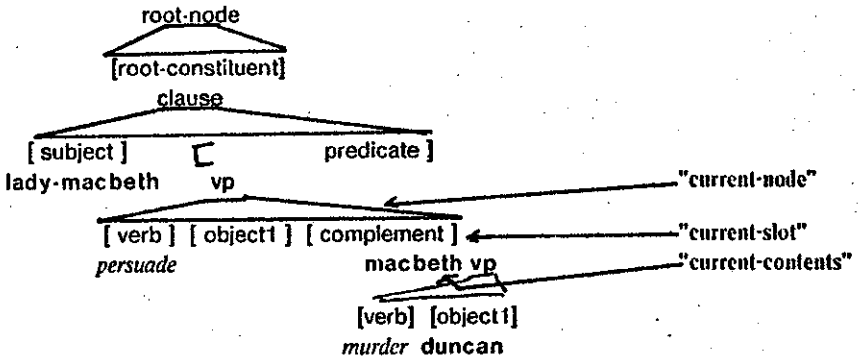
BEFORE	AFTER
Phrase: (basic-clause () predicate (vp-object1 () verb *persuade*))	Phrase: (vp-object1 () verb *persuade*)
Map: ((subj . (subject)) (obj . (predicate object1)))	Map: ((obj . (object1))

Figure fourteen

Transformations do their job by editing a copy of the choice that they apply to. In the case of "equi", the editing removes the dominating clause node from the phrase-schema and the subject constituent from the map, leaving only the schema for the verb phrase to be instantiated as the realization of (macbeth (murder (duncan))).

Subordinated clauses often must be prefixed with a "complementizer", a word from a close class of alternatives including, in this case, "*that*" and "*to*". We can not put the responsibility for this complementizer in the dictionary because we are specifically abstracting the dictionary's linguistic specifications away from the peculiarities of individual grammatical contexts. The responsibility cannot be given to the transformation that does the subordinating, since for theoretical reasons the transformations do not have the ability to manipulate the so-called "function words" directly (see section <2_color_hypothesis>). This leaves only one other place—the grammar routines. Which grammar-routine should be responsible, i.e. what would be the correct constituent structure label and controller event to index it by can be shifted by changes in the design trade-offs taken by the grammar writer; my present analysis (pg.<attachment_points_and_data_types>) is that it should be associaed with the slotname "complement" and the controller event "after-realization".

The last snapshot catches the controller at just this point, after the verb phrase has been instantiated and knit into the tree but before the controller has begun to traverse it.



"Lady Macbeth persuades Macbeth... //"

```
(define-label  complement
    after-realization
        (IF  (equals  'clause (category  curent-node))
            (THEN  (decide-whether-to-say-"that"))
        (ELSEIF   (equals  'vp (category  current-node))
            (THEN  (say "to")) )))
```

Figure eleven

The grammar-routine first checks to see which of the two possible subordination strategies was actually picked.[11] If it was equi, then the grammar-routine has the word "*to*" entered directly into the text without ever occupying a slot or being processed by the morphology routine.[12] If the choice was that-complement, then the grammar-routine must initiate a further decision since the complementizer "*that*" is usually optional. This "decision" is no different in form from the decisions that make up the entrys in the dictionary, except for the place from which it is made. Individual dictionary designers may replace the grammar-supplied decision with one of their own if they wish.

The remainder of this example is straight-forward. The controller moves into the new verb phrase, eventually reassigning the variable current-verb but never current-subject, since it is "managed" by clause nodes only and the clause node that might have dominated complement constituent was stripped away by the transformation. The verb "*murder*" is left in the infinitive because of the rule embedded into the morphology routine making it sensitive to the slot that contains the current verb phrase. The final punctuation—a sometimes non-trivial detail, since it is associated with the definition of a sentence's boundaries and these may not be decided upon until the controller has actually reached the next constituent—is realized when the controller has finally wound its way back up through the tree and "leaves" the "node" that is the "current-sentence".

2.6 Relation to previous work

Two other perspectives have been taken in other computational studies of production; they are discused at length in another paper [t_past]. Briefly, there is one school that we could term "grammar-controlled linearization and translation" [simmons_&_slochum][goldman] [wong] [heirdorn_best_production_ref][shapiro_75][shapiro_79], and another, larger if less linguistically sophisticated school that we could term "production directly from program data" [shrdlu] [Swartout] [mycin_production_reference][chester][roberts_frl_generation]. (Two other important systems: [erma] and [davey], fall into neither of these categories as they both employ extensive grammars and vest control into non-grammatical processes; unfortunately, neither has been further developed.)

---

11. It might be "safer" to couch this test directly in terms of the names of the alternate transformations (which will have been included as part of the discourse history). As it stands, any subordinating transformation that produced a clause could draw the "*that*" even if it was intended as a "obj-inf" construction as in "*Your country expects every man to do his duty*". In the present grammar, a "raising to object" construction such as that must be analyzed as involving two separate arguments at the message-level, the sources for "*every man*" and "*⟨every man⟩ do his duty*" if it is to be handled without changing the existing [complement after-realization] routine.

12. Here we have an example of an indirect constraint by the linguistic component's design on the meaning of its terms. All and only closed class vocabulary words are printed by grammar routines; since the "output" from a grammar-routine goes directly to the text without the possibility of being processed by the morphology routine, this establishes the defacto requirement that any word that undergoes morphological adjustments to its orthographic form depending on the grammatical context cannot ever be a closeed class word—a restriction which agrees with the linguistic concensus.

The grammar-controlled school vests total control of the process in a topdown generative grammar typically given as an ATN. The grammar hypothesizes a way that the message might be realized, tests the message to see if that way would work, then constructs that part of the text if the tests succeed, otherwise backs up and test the next grammatical possibility. Texts are produced as a side-effect of traversing the ATN. Compared with using the message itself to control the process, this technique is inefficient at best. More importantly, this design 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 [Steward Shapiro, personal communication 1979]; I conjecture that the reason for this is that the space of possible message configurations that these systems have daNAlt with is relatively small, allowing it to be encodedd as tests for all the possible contingencies directly on the arcs of the ATN grammar; I predict that when the contingencies become too diverse to anticipate, the grammar-controlled systems will metamorphose into a more message-controlled style.

The "direct production" school is much closer to my own philosophy. Their approach is to start with a data structure of the expert program (their "message") and, in effect, evaluate it with a special "text production" evaluator just as in other circumstances they might evaluate it with, e.g., the normal LISP evaluator in order to perform some function. The structure of the message governs what production processes are run and in what sequence (usually a strict, depth-first sequence, evaluating arguments before functions and using the internal program stack to record what to do next and what to do with "subtexts" as they are constructed). The "production functions" for individual kinds of program objects assemble texts by embedding texts produced for their sub-objects in a matrix text—entirely the same role, conceptually, as played by dictionary entrys in my model. I believe that the difficulties these systems face—almost virtual grammatical naivete and an inability to produce text that is not absolutely isomorphic in structure to its message— could be overcome, I hold, if they were to adopt an intermediate, linguistically motivated representation in terms of which to realize the elements of their messages and which, suitably interpreted, could then serve as a ready description of context and a mechanism for the automatic (i.e. not expressly mentioned in the message) application of general rules (a policy which, not uncoincidentally, is the central theme of the present theory).

3. Micro-Speakers

One cannot develop a theory of a transducer without having some model of the input the process will recieve and the motivations behind its structure. This part of this introduction will present the *micro-speakers* that I devleloped to serve as this model. We will look at the kinds of messages they produced and the texts that my program, MUMBLE,[13] produced from them. MUMBLE is an experimental implementation of the my theory of natural language production and serves as an experimental vehicle for exploring different analyses of linguistic constructinos and the heuristics for using them. Its technical specifications are given briefly in appendix A.

3.1 Why 'micro' speakers?

Compared with the quality of texts being produced by other programs in the literature, this linguistic component has been intended from the outset to facilitate very sophisticated production, e.g. the motivated use of: embedded clauses, ellipsis, pronominal and non-pronominal subsequent reference, adverbial phrases, heavy-noun-phrase-shift, arbitrarily embedded wh-movement, thematic relations such as focus and given/new, and the complete verb group. (These are the contents of chapter three.)

To appropriately utilize such capabilities, an equally sophisticated speaker/expert-program combination is required. However, while the basic linguistic theory to support the design has been available for some time, the operational programs which would act as a test bed have not been. At the MIT Artificial Intelligence Lab the closest to an adequate program was the "Personal Assistant" project, PAL [candy_pal]; unfortunately it was never given a motivated speaker component.

As a consequence of the lack of suitably powerful "real" speakers developed by others, I was forced to develop test-bed *micro-speakers* by myself, with corresponding less ambitious goals since the thrust of this research has been toward linguistic rather than pragmatic problems. To date, five micro-speakers with minimal expert-programs have been experimented with. Each provides examples of different kinds of linguistic problems. Each uses a very different style of representation. (They are all are written in LISP, however, as is the linguistic component.) This variation of representation is very deliberate. It is appropriate because the linguistic *component* is intended as such—one module that is to be combined with other modules written by other designers. I have considered it very important to demonstrate the viability of the linguistic component with speakers employing very different kinds of representations. The results of the experiments with the five speakers are presented as evidence of this.

---

13. This is not an acronym.

In this section, we will look at the three most thoroughly developed micro-speakers: the logic domain, the KL-ONE-nets-as-objects-domain, and the Macbeth domain. We will look a representative samples of their messages and the texts realized from them, and we will take advantage of those examples to point out some the linguistic problems that MUMBLE is capable of dealing with. Below is a brief sketch of all of the micro-speakers that were developed in the course of this research.

The logic domain Here, well-formed formulas in the predicate calculus, are supplied directly to MUMBLE, e.g. from ∀(x) man(x) — mortal(x) we get: "*All men are mortal*". The domain is 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 coherency in discourse, and the symbolic analysis of possible realizations.

KL-ONE nets as literal objects KLONE is a highly structured semantic net formalism under development at BBN [brachman_thesis][current_kl-one_ref]. In this experiment, KL-ONE nets were given directly to MUMBLE to be understood as literal descriptions of "concepts", "roles", and "subconcepts", the primitives of the KL-ONE representation.[14] From one net would be produced a multi-paragraph text—one paragraph per concept, following a depth-first traversal of the net from concept to subconcepts. This domain provided an opportunity to study 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.

Tic-tac-toe This domain is still in the early stages of development, however it already provides useful examples. It is modeled after the work of Anthony Davey [davey], whose thesis at Edinburgh presented a program that gave fluent commentaries of games of tic-tac-toe that it had either played or read. My version employs a object-based representation (as opposed to the representation of the KL-ONE domain, which is relational), and provides an opportunity 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*".

The Digitalis Advisor The work on this domain was done by Ken Church part-time during the fall of 1978. The inputs to MUMBLE were literal procedures taken from DIG, the digitalis therapy advisor developed originally by Howard Silverman [original_dig] and was reimplemented

14. This is as opposed to using the concepts represented by a KL-ONE net as a source of descriptions for individual objects defined in terms of them. This is the intended use of the KL-ONE representation; the micro-speaker has just been a device for producing large texts (i.e. dozens of paragraphs) for the purpose of linguistic experiments without undergoing the overhead of actually developing a speaker program that could independently motivate comparably sized texts. A dictionary for this more natural use of KL-ONE is being prepared but has not yet reached the stage where it can be reported. However, individual examples from this dictionary will be used later in the thesis when they are the best source of examples on a given point.

for explanations by Bill Swartout using the language OWL [owl_ref]. The resulting texts were comparable to, though not quite as smooth as, the texts originally obtained by Swartout in his Master's thesis. 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, MUMBLE is not a good place to stage large-scale reanalyses of a domain's conceptual structure.

An earlier version of MUMBLE (circa. 1976) worked from issolated, hypothetical messages drawn from the Personal Assistant Project, the Blocks World [shrdlu][winston_book], and the MACSYMA Advisor developed by Michael Genesereth [genesereth]. Occasional examples will be drawn from this work when they are appropriate.

3.2 The LOGIC Domain

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

The *logic domain* consists of a representation for well-formed formulas in predicate logic (described in the next appendix), routines for translating formulas typed by a user into this representation and storing them, and a dictionary with fixed entrys for the logical connectives and inference rules and a set of conventions for new entrys 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 entrys of the dictionary.

The original work with the logic domain consisted simply of presenting MUMBLE with a single wff and having it produce an English rendering. For example

$$\forall(b)\ \forall(s)\ \text{space-for}(s,b) \leftrightarrow (\text{table}(s) \vee \text{cleartop}(s))$$

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 the analysis technique described in reasoning_about_possible_choices_n.y.w. and simple tests of the contents of the expressions to determine that the interpretation would go through. The same formula, say:

$\forall(x)$ 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 exaust the linguistic insights to be gained from looking at single formulas in isolation. A predicate calculus formula is vague 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. The accompanying figure shows a natural deduction proof with 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 rendition of the set of all sets paradox.)

---

line1: premis
$\exists x$ (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: prop. calc. (2)
$\forall y$ shaves(g,y) $\leftrightarrow$ $\neg$shaves(y,y)
line4: universal instantiation (3)
shaves(g,g) $\leftrightarrow$ $\neg$shaves(g,g)
line5: prop. calc. (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)))
— (shaves(g,g) $\wedge$ $\neg$shaves(g,g))
line7: prop. calc. (6)
$\neg\forall x$ (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 Giuseppi. Now, anyone who doesn't shave himself would be shaved by Giuseppi. This would include Giuseppi 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.

The lines of the proof are passed to MUMBLE in sequence. The rendering selected for earlier lines provides a discourse context to narrow the choices available to later ones for subsequent

references to constants, variables interpreted as generic references, and predicates and formulas used as descriptions. Further motivation is provided by the labels that can be attached to certain lines to reflect their role in the structure of the proof: "*the assumption*", "*a contradiction*", and by the 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 Dan Chester [chester] 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 relatively rare in studies of language production. Chester's version of the "barber proof" is as follows:[15]

> *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 used the common technique of recursive replacement in conjunction with an ad-hoc grammar. (Comparable techniques were used in the explanation systems developed for MYCIN [mycin_explanation_ref], for Swartout's Digitalis Advisor [swartout] and for Weiner's BLAH system [weiner_thesis].) The key difference between this technique and my own is its lack of any explicit representation of the linguistic structures it is producing: this is reflected in Chester's minimal treatment of subsequent reference and the occasional abruptness of transition from line to line (i.e. his program has nothing to look at to tell it that the conclusion of one line is not obvious from earlier ones). A more extensive description of the differences between our two programs is given in [ddm_past].

At this point, I will use the example of the barber proof to point out some of MUMBLE's accomplishments.

The ability to go beyond the literal content MUMBLE processes a proof by scanning its formulas in order top down, starting with the inference rule for the line and ending with the individual variables and constants.[16] The contents of 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) a logical connective is to be realized. and which (if any) of the subelements of the present formula are to be involved in that realization. Line three has no corresponding sentence

---

15. My source for Chester's results is a personal communication with him in November of 1975; the major effort on the logic domain for MUMBLE was completed in December of 1977.
16. In other words, the enumeration order used by the logic domain's entrys (pg.<message_element_enumeration_order_n.y.w.>) is dictated by the construction axioms of the predicate calculus.

because we can assume[17] that that step in the proof would be automatic to any audience. Line four, on the other hand, has been expanded into three sentences because the 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 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 construction of this argument is discussed on pg.32.)

The logical conjuction of line one is interpreted as a conventional way of denoting the type of the variable "x"; similarly the two quantifiers of that line are realized in the determiners of their variables ("*some barber*", "*everyone*") rather than as "*for*" phrases. (Chester does this some of the time.)

Subsequent reference Knowing when *not* to use a pronoun is very important to 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 (section focus_n.y.w.) is on the universally quantified variable "y" rather than "*Guiseppi*". When the focus shifts to him in sentence five as a result of the use of the intensifying reflexive ("*Guiseppi himself*" he can then be pronominalized in his four instances of sentence six.

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*". (Both are discussed in section IV.C.3.1.)

Functional labels The premis 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 $A \wedge \neg A$ 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 (line eight). 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 the

17. This is an implicit, conventional assumption: there is no simulation model of the user.

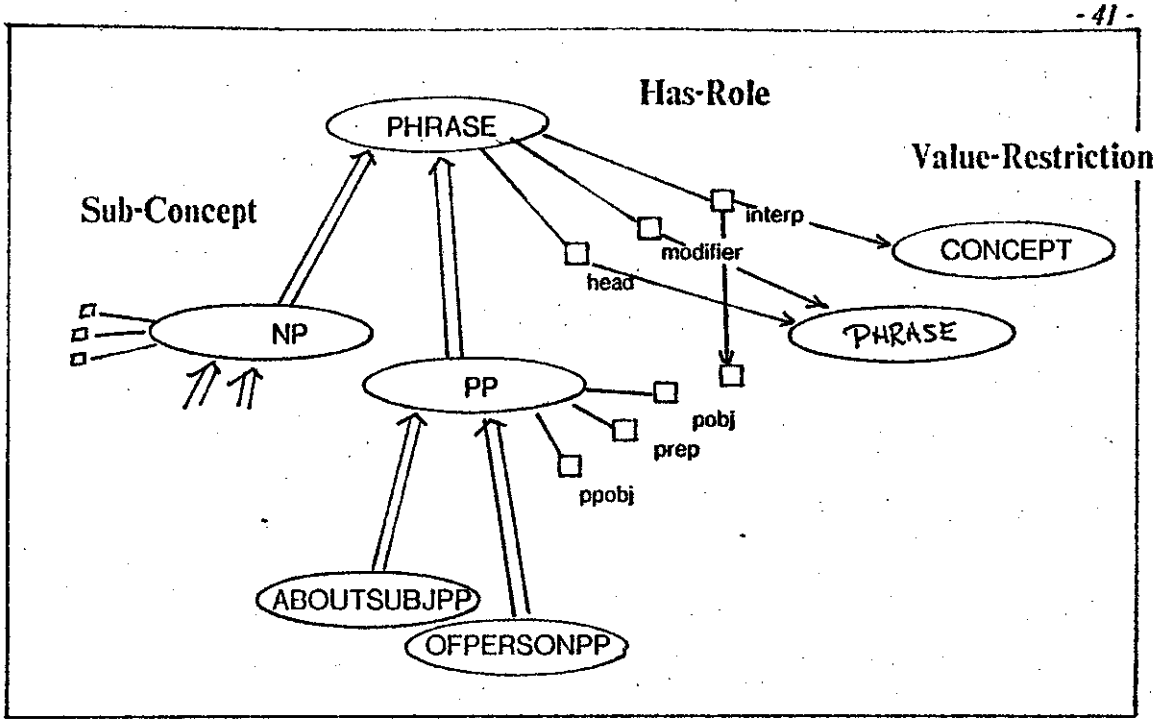
decisions for the output text is a rhetorical description of the discourse structure. The different terms of this structure will guide decisions at syntactic and morphological levels: in sentences one and three a contraction is used ("*doesn't shave*") while the same logical structure in the *formal* context created by the *conclusion* sentence of the subargument is not contracted. 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.

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 just the distance between phrases: when, as in line five, a contradiction is deduced from the immediately previous line, the identification of that deduction is given in the most direct way possible by adjoining a relative clause to the last sentence (sentence five); when the dependency line is much earlier (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 elects to say "<restriction> *and* <complement of restriction>". (An equivalent technique would have been to replace the universal quantifier in "*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 ambiguites in the conjunction. It notes the point where the conjunction is attached (i.e. at 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, and then decides that it is possible that the second arm will be missinterpreted as conjoining with the more immediate, lower direct object rather than to 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.

3.3 KL-ONE-nets-as-objects

A KL-ONE net consists of a set of named objects of various types (only concepts and roles are shown here), linked together by the relations: subconcept, has-role, value-restriction, and others not shown. The KL-ONE-*nets-as-objects* domain takes such a network as input and produces 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...")).



The utility of this domain has been largely as a means for me to become familiar with the KL-ONE language in preparation for developing a dictionary and interface for a speaker based on "real" KL-ONE (some of the intial results for which appear in section <dictionary_for_real_kl-one>). Beyond this, the KL-ONE-nets-as-objects domain was my first experience in producing multi-paragraph (if boringly uniform) texts, and it is potentially useful in itself as an alternative "read-out" mechanism for use in debugging KL-ONE networks.[18]

The figure shows the first paragraphs of the text constructed for one of the development networks in use at BBN during the spring of 1979: 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.)

The four and a third paragraphs shown in the figure are sufficient to illustrate the stylistic heuristics that the dictionary for this domain incorporates. (Like the logic domain, KL-ONE-nets-as-objects has no speaker as such; its messages are comprised directly of KL-ONE nets or coherent

18. This mechanism has not yet been built. Two things have stood in its way: one is the fact that KL-ONE is in a constant state of redesign and is thus a "moving target"; and the other is that the chosen implementation of KL-ONE has made it awkward to treat the basic relationships of a KL-ONE net as first-class objects because of the large amount of "meta-level" structure required (see pg.<artificial_first_class_objects>).

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]]

subnetworks.)

Varying the paragraph structure In each of the first three paragraphs, the presentation of the concept's roles and their value-restrictions is given in a different style. Is is done by having the concept-defining-entry (pg.<concept_defining_entry>) vary its choice according to the number of roles there are: 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 such places that the concept-defining-entry assumes that it will be still remembered. Similarly in the second paragraph, where there is a summary sentence listing pp's roles, the has-role facts have been left off of the later sentences. Not doing that would have led to a noticeably redundant text (see section <given/new>).

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 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). The entry for roles appears on page <role_entry_n.y.w.>.

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 from their content (see section <ellipsis>).

ump_state "ax_2"

The "Macbeth domain" is the newest of all the micro-speakers. The representation it uses (frames, "almost English" assertions) is a common one for present day AI programs, suggesting that its treatment by MUMBLE may be typical of what could be done for the many programs like it. Consequently I will discuss this domain in greater detail than the previous two.

3.4 The 'Macbeth' Domain — an example speaker and expert program

Patrick Winston has developed a program for making and evaluating analogies [winston_analogies]. It takes two descriptions, represented in a variant of FRL[19] ("Frame Representation Language"), and establishes a correspondence between them as for example between the laws of fluid flow and ohm's law. Winston is interested in how differences in the vocabulary of the descriptions: choice of property names, the availability of an "a-kind-of" hierarchy or of one-step inferences, the use of backpointers, and so on, effect the ability to make analogies and generalizations based on them.

Winston has written FRL "synopses" of several of Shakespeare's plays in terms of *frames* for the major characters and scenes. By considering facts such as whether a king is murdered or whether there is a marriage, the program will conclude that "Macbeth" and "Hamlet" are more alike than, say, "Macbeth" and "The Taming of the Shrew".

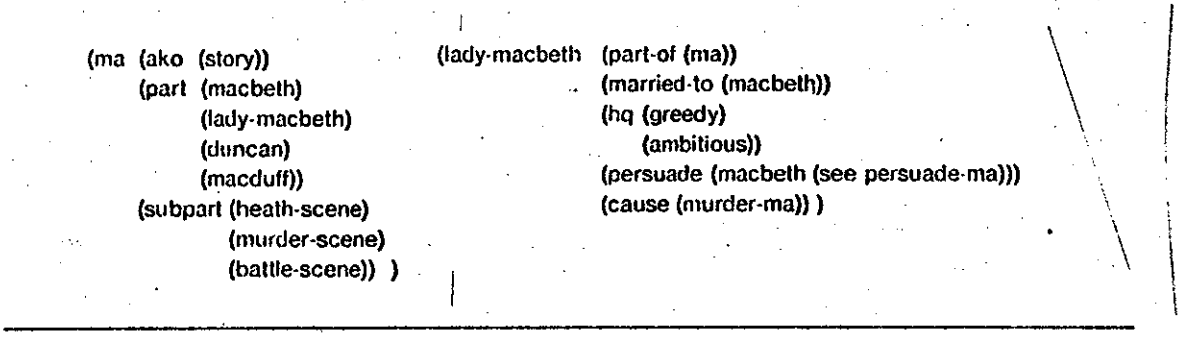
From FRL to English

At the moment, Winston's program presents its results as tables of correspondences and confidence values. Ultimately, we would like these presentations to be in English, and to include plot summaries and explanations of the program's reasoning. This will require the development

19. FRL was developed by Roberts and Goldstein [frl_manual][frl_primer] to explore some of the ideas in Minsky's 'FRAMES' paper [minsky_frames]. Unlike other "frames" systems such as [krl_paper] which are entirely new programming languages/environments, FRL is a relatively uncomplicated, direct extension of the property-list mechanism of LISP.

of a *speaker program* to bridge the gap between Winston's program (which knows nothing about natural language) and MUMBLE with its knowledge of English grammar and production constraints.

A proper speaker for this domain has not yet been designed, though the first necessary step, the capability to go from the analogy program's data structures to individually coherent English texts, has been taken. This involved the design of a set of *interface functions* that interpreted the FRL representation for MUMBLE and a minimal *dictionary* that MUMBLE used to interpret individual frames, the common analogy vocabulary, and the special vocabulary for specific topics. A dictionary for Winston's synopsis of "Macbeth" was developed in July of 1979 in approximately one week. The figure shows the frames for the synopsis of "Macbeth" (abbreviated MA) and for the character "Lady Macbeth".



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

(lady-macbeth  (part-of (ma))
               (married-to (macbeth))
               (hq (greedy)
                   (ambitious))
               (persuade (macbeth (see persuade-ma)))
               (cause (murder-ma)) )
```

Every frame is a LISP list and consists of the name of the frame followed by its properties, each of which is itself a list consisting of the name of the property (e.g. ako—"a kind of", or hq—"has quality") followed by its argument(s). If a property has more arguments than just one, or if it is referred to by some other property, then it is represented by a frame of its own, e.g. murder-ma, persuade-ma (given on pg.<murder_ma>)

As you can see, these frames are designed to already be quite close to English in their structure and the level of their conceptual vocabulary. Each property of a frame already has the approximate form of a declarative English clause, i.e.

$[_{clause}\ [_{subject}$ ma$]\ [_{verb}$ako$]\ [_{object}$story$]]$

Because of its simplicity, the minimal dictionary for the Macbeth domain needs to have only two entrys:

whole-frame_entry: — builds a paragraph about the frame, mapping each of its properties into a separate sentence.

20. i.e. the name of the frame (e.g. ma) becomes the discourse focus of the paragraph; see below.

default_property-entry: — maps the components of a frame property into the constituent positions of a simple clause. (A listing of this ENTRY as used in MUMBLE appears on page <default_property-entry>.)"

These two basic ENTRYs are then supplemented with trivial, schematicized ENTRYs for the particular vocabulary of the Macbeth domain, which allow us to, e.g., say "*I is a character in*" in place of the FRL property part-of (see pg.<part-of entry>).

Given this much of a dictionary, we can produce texts like the one below, which was created directly from the frame MA shown above. has handled all of the grammatical details (agreement, sentence-initial capitalization, commas inside lists, determiners for the predicate nominatives, and colons for the list apposatives) and has heuristically applied stylistic transformations to make the text more readable (conjunction reduction—pg.<conjunction_reduction>, and "predicate merging"—pg.<predicate_merging_lexical_version>).

"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.

This ability to take combinations of simple propositions and realize them as a fluent, coherent text is possible only because of the extensive linguistic description that MUMBLE brings to those propositions which makes it possible to apply general purpose rules to the descriptions themselves without having to know anything about the propositions.

The special job of a 'speaker'

The speaker is the "mouth-piece" of the expert program. Its function is to communicate selected information and intentional attitudes of the expert to the human user, which it does by passing expressions ("messages") to MUMBLE. Messages are descriptions of the speaker's goals, interpreted in terms of English texts. In many cases, the simplest description of the information to be communicated is the same data structure that the expert uses internally to represent it. However, that is not always true, and this is where the special position and knowledge of the speaker come into play.

The representational priorities of an expert program are seldom the same as those of people using natural language. Both will refer to the same information, yet, because it is used to different ends, the information's form, degree of redundancy, and modes of access will be different. For example, it is convenient for the analogies program to maintain redundant backpointers between frames. The last two properties of the lady-macbeth frame (repeated below with with the referenced frames spelled out) are an example of this as each points indirectly at the other.

(lady-macbeth (persuade (macbeth (see persuade-ma))) (cause (murder-ma (see cause-ma))))

(murder-ma (backpointer (frame (macbeth)) (property (murder)) (value (duncan))) (caused-by (lady-macbeth)) (coagent (lady-macbeth)) (motive (ma-become-king)) (instrument (knife)) (time (murder-scene)))

(persuade-ma *;backpointer omitted* (purpose (cause-ma))) (act (murder-ma))

(cause-ma *;backpointer omitted* (method (persuade-ma)))

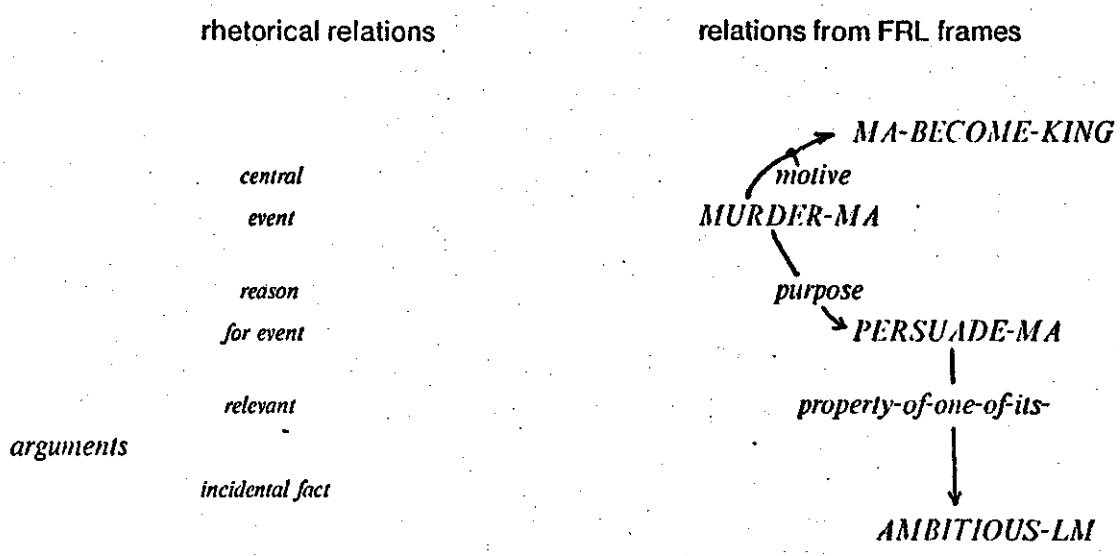
While this redundant representation is very convenient for the analogies program (it makes it easy to trace association links through the frames using a marker-passing style search), if taken as a literal description of what to say, it leads to a very awkward English text as shown below:

> *Lady Macbeth persuades Macbeth to murder Duncan in order to cause Duncan's Murder. She causes it by persuading him to do it.*

The text is awkward because its source message is not appropriate—it does not follow the expected conventions. When using natural language, you do not feed the audience facts as if you were loading a program. People react to each item as they hear (read) it, and expect to be continually adding to a coherent picture according to a discernible chain of reasoning. (Hence the problem with this "literal" text: the second sentence has no plausible motivation.) Order of presentation is important; what is explicitly said versus left to be inferred is important; and the foregrounding and backgrounding of individual items is important—natural language is a sequential rather than a relational medium and it must be actively parsed; it thus requires adherance to conventional signals if the audience is to have an easy time of it.

The job of the speaker is to understand how the representation of information and intentions in a text and in a (particular) computer program differ and to formulate messages which bridge the gap. The expert's data structures can still be used as the raw material of messages, except that now they are embedded in a matrix of relations that are special to natural language based communication. The expert's facts have been "cut and pasted" into a new expression based on different representational principles.

The figure shows how the (hypothetical) rhetorical schema on the left would be instantiated by individual frame properties (on the right) in order to make a point about the play (i.e. why Macbeth murdered Duncan) in a way that an audience would understand.



In addition to this "propositional content", the conventions of English require the speaker to decide in what order the propositions are to appear, how they relate to each other, and what temporal point-of-view to adopt. Furthermore, since we are constructing a discourse, we must specify its focus—what it is "about". The message below conveys all of this information, and is a good example of the sort of message that MUMBLE has been designed to handle. (This message has the form of an FRL frame. This is convenient for this domain but not essential.)

```
(message (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)) (ancillary-facts ( (murder-ma (motive (ma-become-king))) ) (
(persuade-ma (purpose (cause (murder-ma)))) ) ( (lady-macbeth (modifies (ambitious-lm))) )))
```

Given the present dictionary, MUMBLE realizes this message as:

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

The reader should appreciate that this example message was assembled "by hand" for the purpose of exercising MUMBLE's capabilities. Its function is to illustrate what could be done—to show where MUMBLE is expected to begin its part of the production process. The technical details of constructing such a message can be subsumed by schematic subroutines and should not get in

the way of the decision processes once the speaker's job has been understood sufficiently to formalize the principles behind a message like this and to permit the design of a serious speaker program (as opposed to the mini-speaker experiments undertaken thus far)

4. When should MUMBLE be used?

Preparing an interface between a new expert program and MUMBLE is not a trivial undertaking. Creating the dictionary will require the designer to provide explicit representations for relations that are left implicit in many expert programs. Thought must be given to the mechanics of message construction and to the practical exigencies of dealing with another independent 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 pleasant looking "linearizations" of them, then it is questionable whether they should go to the effort of using MUMBLE as there is already an established technology for producing "English-like" output from well-structured data that requires considerably less overhead (see for example [roberts_frl_to_english]).

MUMBLE's strong point is its ability to combine disparate data on the basis of linguistic descriptions to produce cohesive, context-sensitive texts. If, to choose an extreme example, all of the remarks that some program would ever have to make could be anticipated at the time the program was written, then there would be no point in using MUMBLE to produce them; on the other hand, if the 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 MUMBLE (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 MUMBLE.

- ☐ Embedded clauses: Any 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 syntax rules coordinating adjustments to the text of the relation and to the matrix 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, and the deliberate use of ordering or explicit relational connectives

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. What the corresponding linguistic choice should be (e.g. noun phrase, clause, or verb phrase) often 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.
- **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 so that only grammatical phrases are built. Planning is required to judge how thorough a description must be and how that will effect grammatical structure, for example whether one can say: *(I put an X on) the adjacent corner* or must say: *...the corner adjacent to the one you just took.*

5. What MUMBLE can't do

Efficiency has its price. In this case, it is intrinsic restrictions on the capability of the linguistic component. These are taken not as a failing, but as the necessary result of a deliberate distribution of tasks according to what components are most suited to performing them. I will claim that the bulk of what this linguistic component cannot do can be done better by other the components that it will interact with.

- **Creative expression—fitting old words to new situations:** MUMBLE does not know what words mean. From 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. (How could it if it is able to be used with expert programs with different conceptualizations?) Dictionary ENTRYs select words reflexively according to precomputed possibilities. In particular, they do not use any sort of pattern-matching on "semantic features", both because of the computational expense and because they are unlikely to ever be refined enough to select individual 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. MUMBLE 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 (the sort which is natural to parsers) to detect 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, at the linguistic-level this possibility cannot be foreseen 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) or, by planning the message in terms of "interlingual" predicates like modifies and focus, potential awkwardness can be foreseen and planned around by general rules.

- ☐ **Reasoning through trade-offs caused by limited expressibility:** It can happen that the inability to simultaniously express, e.g., modality and subordinateness, will not become apparent until the realization of the message is already begun. For MUMBLE to be able to reassess the relative importance of the message elements that prompt those aspects (1) it needs 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) it needs to be aware of the potential problem early enough to be able to plan alternatives (see pg<alternatives_to_island_violations>). Without such a vocabulary, MUMBLE must rely on the ordering provided initially in the message and the speaker must be prepared for not all of its messages to be realized.
- ☐ **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 MUMBLE to perform the ends-means reasoning required to bring it about, e.g. one cannot tell it: "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 and a high processing overhead. On the other hand, in writing the dictionary the designer could precompute the decision-space such a goal would entail and then incorporate it into MUMBLE as a set of rules in the grammar. (That goal is roughly equivalent to the existing focus heuristic for example.)

Additionally, MUMBLE has no analysis for some English constructions simply because none of the micro-speakers were able to motivate them. These include productive compounding of pre/suf-ixes, comparatives, and quantifier scope.

6. Intellectual roots

The initial seeds of this research were planted while working with Terry Winograd on the SHRDLU program [shrdlu]. The basic linguistic principles of the present design: the organization of the grammar around systems of choices, basing descriptions on feature systems, and the general approach of viewing natural language as a tool used to perform a communicative function were also part Winograd's work and are emphasized by the theory of *systemic grammar* on which it was based.

My undergraduate training was in linguistics at MIT. The influence of transformational generative grammar is evident in my choice of linguistic vocabulary, and, more importantly, in a concern for "capturing generalizations" and a metatheoretical bias that the constraints on a process should follow inescapably from the structure of the device which implements it.

It is harder to properly attribute the computational principles of the design: data-driven programming, procedural attachment, symbolic description, and the extensive use of interpreters and schematic data-structures. Much of it is an outgrowth of the programming style encouraged by LISP [mccarthy_recursion_paper]. Other specific influences are Hewitt's PLANNER [microplanner_manual], the work of Sandewall [sandewall_biblio_paper], that of Sussman [art_of_the_interpreter], and of course the incredible atmosphere of the MIT Artificial Intelligence Laboratory as a whole.

This research does not derive from any of the earlier direct work on language production either in Artificial Intelligence or psychology, although some of the recent design details have been inspired by psycholinguistic findings of Merrill Garrett [three_garrett_papers].

7. Contributions of this thesis

UMBLE is the most linguistically competent natural language production program that has been written to date. This is due primarily to the advances in the computational theory of production reported in this thesis, which have simplified the process of representing linguistic rules and usage-heuristics. This is the first theory to be specifically designed for use with programs with different representational systems.[21] This is the first theory to be grounded on pycholinguistically plausible hypotheses such as left to right refinement and production of text,[22] a limited working buffer, and indelible decisions. Specific improvements over earlier systems include:

- ☐ 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 effects to be achieved by the text.
- ☐ The linguistic structure of the text being produced is explicitly represented.[23] 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.
- ☐ The possible realizations of each element of a message are explicitly represented and are available for inspection or special-case manipulation.

---

21. The atn-based generator originally developed by Simmons and Slochum [simmons_&_slochum] and later adopted by Goldman [goldman] has been used with many different programs: [margie][lehnert_thesis] [yale_report_w_chinese_generator][meehan_thesis], however, all of these employed the same representational system: conceptual dependency [schank_good_cd_ref].
22. Gerard Kempen [kempen_march_1977] writes that production should be incremental and left to right, however, his program as described [Kempen_1979] actually refines the constituents of each clause in parallel.
23. This was true also of the German-to-English translation program of Gretchen Brown [gretchen_thesis], and locally true in the systemic grammar used by Anthony Davey [davey].

CHAPTER TWO

DEFINITIONS

In this chapter we will look at the basic algorithms of the theory and at the types of objects that they manipulate. These are the building blocks out of which specific grammatical and stylistic rules will be fashioned, and their properties will determine the limits of the production process's capability. Specific analyses may change as our understanding of them grows, but the algorithms and data types of this chapter will not change, or at least not drastically, unless the theory is found to be intractable.

1. Terminology

Throughout this paper, I will distinguish between "language production", the "linguistic component" and the "program MUMBLE". *Language production* is the activity which ensues between the time when the speaker (human or machine) first realizes s/he has something to communicate and the utterance is completed. The phrase is to be a pretheoretical term, which, in particular, does not presuppose how the process breaks down into subprocesses or how the use of linguistic decision-criteria is distributed.

The *linguistic component* is the subject of this thesis. I am proposing an analysis of the production process wherein all of the language-specific knowledge is resident in one subprocess, the "linguistic" component, which operates after the content of the utterance has been decided upon and does the actual production of the text. The linguistic component will be defined in terms of an abstract specification (the subject of this chapter), whose semantics will ultimately be grounded in terms of LISP data types and operations.

The LISP *program* named MUMBLE is a particular instantiation of the linguistic component for which a specific English grammar and various dictionaries have been written. The grammar and dictionaries are defined in terms of the theoretical vocabulary of the linguistic component, but they derive from analyses that, while designed for that component, are not the only conceivable ones which could be used.

The linguistic component functions as one component in a larger system that includes at least (1) an *expert program*—presumably the reason why all of the programs are assembled together in the first place, the other programs being just an extended "user interface"); and (2) the *speaker* who formulates the messages to be communicated to the user.[1] In this paper, the speaker and expert program together will sometimes be referred to as *the domain*, as in "the logic domain" or "the Macbeth domain". This term is intended to focus on the contrast between linguistic and non-linguistic representations or operations, the former residing all the the linguistic component and the latter all in a domain.

Two classes of human beings have dealings with the system: designers and users. *Designers* write the programs that make up the system. *Users* are the people for whose benefit the system is built—the people with whom the system will be communicating. Designers may be further subdivided into *dictionary designers* and *grammar writers*: grammar writers must usually be sensitive to the decisions of the dictionary designer (see for example pg.<ontological_differences_between_domains_n.y.w.>).

1.1 Properties common to all types of objects

The rest of this chapter is devoted to specifying the individual characteristics of the data types of the theory. This section will discuss what they have in common.

The linguistic component's fixed procedures manipulate individual *objects*—tokens of the defined data types. The term *object* will be used throughout the paper as a place holder whenever data-type specific properties are not relevant to the discussion, or when the intended type(s) are deliberately being left vague. Just what objects will exist in any one instance of the linguistic component (such as MUMBLE) will be a matter of what speaker and expert program the component is working with and what grammatical analyses are employed. On the other hand, what data types there are is fixed as a matter of language production theory.

The linguistic component is an independent process within the larger system. While it is only active for the time during which it is realizing messages, its computational state is maintained across the dormant periods. Against this background, we can distinguish temporary from

1. The "complete" system would presumably include a *language understanding component*, which should share its description of English grammar the linguistic component and must share its record of past discourse. However, I will have nothing specific to say about the relationship of the two components in this paper

permanent objects. *Permanent objects* make up the grammar and dictionary. They represent facts, relations, etc. which will not change from message to message. *Temporary objects*, on the other hand, are the stuff of which the linguistic component's working structures and context are made. They are typically *instances* of generic, permanent objects, created as needed at specified points within the process and later explicitly expunged when the information they record is no longer needed in the same detail (see section <expunging_temporary_objects>).

Objects are to be understood as bundles of properties. More formally, each object is defined to be a *vector* of other objects (the recursion terminates on objects of type SYMBOL), and *properties* are selector functions against such vectors. Two properties in particular, data-type (occasionally abbreviated to "type") and name, must be included with every object. (Accordingly, they will be omitted from the individual data-type specifications.) The data-type determines what other properties the object may have; the name determines its uniqueness, i.e. objects of the same type will be seen as identical if and only if they have the same name. Data-types will be used to two ends. The first is primary: every object used by the linguistic component has a data-type that defines its properties and large-scale behavior (see II.3.xi). The second is expository: several "syntactic" types will be defined to aid in grouping objects (each with its own "primary" data-type) in terms of the operations that create them or manipulate them.

The possible *values* of each of an object's properties are restricted to be objects of some specified data-type(s), this specification being the principle part of the definition of the object's own data-type. Most of these types will be defined within the linguistic component, but others, such as name or data-type, will be taken as primitive and defined only at the implementation level. For example, in the MACLISP implementation of MUMBLE, objects are implemented as "atoms", and their names as "pnames"—strings of ascii characters. Properties are either fields in records (if there are a fixed number of them) or tag—value pairs on the atom's property list.

Needless to say, specific operations are included in the linguistic component to create and expunge objects and to set, interrogate, and delete their properties both individually and by class. A listing of these operations will not be included with the definitions unless the discussion requires it; for example, classes of expressions are occasionally given a data-type in order to single out a class of operations with a common domain and range. The interpreter(s) that evaluate such expressions will be taken to be primitive. When there is a need to identify a specific interpreter—usually because of the special computatational environment in which it does its evaluations—it will be referred to by a specific name. Just as some of the data-types are primitive, the syntax of expressions involving them will not be relevant to the theory, but only to specific implementations such as MUMBLE and its specific interpreters. When there is occasion to refer to expressions in this text, LISP syntax will be used, i.e. parenthesized lists where the first item of each list is the operator and subsequent items are its arguments:

<expression> ::= (<operator> <argument>*)

In terms of program structure, a designer will see MUMBLE as a series of files containing the primitive operations and basic procedures defined as LISP functions, followed by files defining the grammar and dictionaries. These consist of calls to object-creating macros, one for each object to be defined. At *system load time*, these macros take the schematic structures written by the designer and expand them: filling in defaults, creating functions, converting to more convenient machine formats, and accumulating cross-indexing information. This is augmented by a *postprocessor* which runs after all of the files have been loaded and takes notice of information that cannot be decided earlier.

Contrasts between computations performed at load-time versus those at run-time are intentional and reflect more than just the vagaries of using computer programs. Computations at load-time are a one-time expense that can be ignored later when the system is actually being used. Generally speaking, run-time computations are optimized for efficiency in time at the expense of a large memory requirement to contain precomputed facts and redundant run-time records.

2. The Control Structure

Unlike any of the previous language production programs that incorporated a grammar, this linguistic component is controlled directly by its input message. It takes no actions, tests no predicates, nor builds any structures that do not immediately forward the realization of that message. Furthermore, none of its actions are redundant; each one will contribute to the form of the final text. Its ability to do this rests on the use of a *data-directed control structure* to dynamically configure the component's actions with each new message.

Data-directed control In a system that employs data-directed control structures (also known as "syntax-directed"), there is no predetermined execution sequence. The set of all possible execution steps is predetermined, of course, but the steps are embodied in independent, unit programs that are not connected by any fixed sequencing links. Instead, the sequencing information is provided by the context. Each unit program is associated with a token or pattern of tokens that can appear in the "directing" data. The overall configuration of tokens in the data determines the execution sequence of the program. This association between data token and unit program is maintained by a *controller*, a relatively small, fixed procedure that controls the order and sometimes also the environment in which actions occur. The controller scans the directing data, following a predetermined path. When it reaches a data token that has (or may have) an associated action, the action is looked for, and if found, accessed and executed.

The key to understanding the design of this program is understanding the controller's algorithm: what is its directing data? what are its specific reactions in response to data tokens of different types? And how are these reactions related to each other temporally and computationally? Such is the subject of this section. First we will look at the directing data, the constituent structure of the tree. Following that we will go through the properties of the controller, including a complete flowchart of its algorithm.

2.1 Constituent Structure

The constituent structure of the tree is distinct from the grammatical labels that annotate it. Constituent structure defines the route that the controller follows as it traverses the tree. It defines "positions" rather than grammatical properties, grammatical properties accrue indirectly to it to the extent that grammatical labels are attaced to individual objects of constituent structure.

Constituent structure is made up of two kinds of objects, nodes and slots. These are temporary objects and come into existence only in the context of the tree as a result of the instantiation of of phrase-schema that are part of choices selected in the realization process. Below are the definitions of the nodes and slots, i.e. a listing of their properties. A diagram of a reasonably complicated constituent structure will follow, after the syntax of phrase-schemas has been discussed.

Data type: NODE

NODEs can be thought of as instances of grammatical categories. Their primary function, however, is to define a subtree from the node to the fringe of the tree over which region-features associated with the node will be in effect, and to define (via its associated slots) a list of "immediate constituents" that will be important for their grammatical relationship to the category associated with the node. They have the following properties:

category A category as defined by some constituent-schema from the grammar (pg.<constituent_schema>), i.e. one of the usual descriptive terms: *clause*, *np*, *pp*, etc..

features A list of category-features or region-features, including at least the name of the constituent-schema that was used to instantiate the node and the node's category. A category-feature describes properties of the node itself, while A region-feature describes properties of the entire subtree that the node dominates. See section <categorys_and_slots>.

immediate-constituents A list of one or more slots.

hooks A property-list used to hold extra-constituent information, such as *tense*, *aspect*, or *modality*; see section II.4.5.

Record A record (pg.<discourse_history>) listing the gammar-variables bound at

the level of this node and the values they had. It serves to represent the node in the discourse history after the actual constituent structure has been expunged.

Data type: SLOT

The immediate constituents of a node are organized in terms of a sequence of labeled positions: objects of type slot (as in "constituent slot"). The constituents *per se* are contained in the contents properties of the individual slots. Slots can be thought of as instances of slot-names. They have two obligatory properties and one optional.

slot-name A slot-name such as "subject" or "adj-complement".

contents One object whose type may be either "node", "elmt-instance" "word-instance" (which includes traces), or else the constant symbol "empty".

features A list of slot-features and/or region-features at least including the slot's slotname

Data type: Phrase-schema

Phrase-schema are used by choices in the dictionary entrys to specify the constituent structure that is to be instantiated if the choice is selected. They are permanent objects and are conceptually part of the grammar even though the are only associated with the dictionary. Phrase-schema may be defined in issolation and given a name and in that way can be shared among several choices.

Phrase-schema do not have properties *per se* but are expressions that are defined in terms of the following grammar.

The grammar of phrases

PHRASE ::= (CONSTITUENT-SCHEMA (CATEGORY-FEATURE*)
{ **hooks** <hook-assignments>}
<constituent>*)
<constituent> ::= SLOT-NAME { **features** <additional-slot-features>} <contents>
<contents> ::= { WORD | PHRASE | TRACE }
<hook-assignments> ::= ((HOOK { ACCESS-EXPRESSION | 'SYMBOL })*)

This is a context free grammar. Non-terminals are given in lowercase in the normal text font. Type names in SMALL CAPITALS may be replaced by any object of that type. Words in **san-serif** are literals. Parentheses, **()**, are also literal. Kleene star (*) indicates that the expression it follows may appear zero or more times without limit. The plus (+) requires the expression to appear at least once. Expressions bounded by curls ("{}") are optional. When curls bound a series of items set apart by vertical bars ("|"), those items constitute a choice set from which one and only one of their number is to be picked.

Phrases are constructed recursively in units of a non-terminal node plus its immediate constituents. These units are called "constituent-schema" and they are defined as part of the grammar. Below is PHRASE that was used in the Macbeth domain.

```
(define-phrase  is-a-character-in
(vp_predicate-nominative  ()
      pred-nom (regular-np  ()
                  head  character
                  qualifiers (regular-prepp  ()
                                    prep  in ))) )
```

Phrase-schemas are typically very "sparse", specifying explicit values for only a few of the total set of slots that their constituent-schemas will create when instantiated. This sparce-ness is natural since phrase-schems function as abbreviations, describing what constituent-schemata are to be combined, what words are included and where, and defining a set of "gaps" given implicitly by the SLOT-NAMEs they do not mention. Phrase-schema are used a means of defining idiomatic phrases broadly construed (see [becker] for an excellent elaboration), in such a way that they can be specialized for different grammatical contexts. In the "is a character in" schema for example, the number of the verb and the predicate nominative constituent have not been fixed, it could be modified by adverbs or modals, or part of it suppressed by ellipsis. All of these actions would be brought about automatically (i.e. without designer intervention) by grammar-routines associated with the grammatical labels in the schema or in the context it is inserted into in the tree. When the phrase-schema is instantiated, the "gaps" left where a constituent-schema specified a slot that the phrase-schema didn't mention are filled with elmt-instances selected by the choice in which

the phrase-schema appears.

2.2 THE Tree

Functionally, the tree is a representation of what the linguistic component has done in the past and the plan of what it will do in the future. Formally, it is a structure comprised of NODEs and SLOTs, Graphically, the tree will grow down from its root node and from left to right in the usual manner for syntactic trees in the linguistics literature.

The root NODE of the tree is constant throughout the operation of the linguistic component (i.e. it is preserved across successive messages). It does not have any linguistic significance and is just a way to collect the trees of successive messages into one tree for technical convenience. It has the name *root-node.* Messages are introduced as immediate constituents of *root-node*—contents of *root-constituent.*

There is no general facility for walking through the tree to examine its contents. This prohibition is enforced by the design of the tree itself, which is threaded only from nodes to their constituents and from left daughter to adjacent rightward sibling. That is the route that the controller follows; no other direct associations between parts of the tree need to be maintained.

Only two locations within the tree are continuously accessible. They are "the present position of the controller" (see below), and "the next available SLOT for a message" (defined on page <arbitrary_length_schemas>). Any others, for example "the place where the last instance of (murder duncan) was positioned" or "the locations of all noun phrases whose determiners are *the*", are not defined by the tree or the controller themselves. That is, there is no general facility for going from an ELMT-INSTANCE, or a CATEGORY to a location or set of locations in the tree. If some such locations are important in the linguistic or pragmatic analysis, they will be either expressly remembered or, more likely, a description of the relevant aspects of the location will be compiled at the time when the controller is at that location and the location itself forgotten. Grammatically important locations in the tree are always relative to the current position of the controller.

2.3 The controller

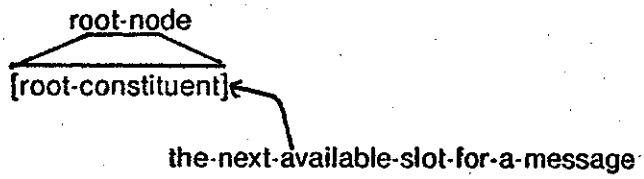
The algorithm for the controller is given over the next several pages. This flowchart covers its initialization and its the three subroutines: process-node, process-slot, and dispatch. Included are the locations of the "controller events" to which grammar-routines may be attached, the points where controller-variables are set, and the location of calls to the realization procedure and morphology routine.

INITIALIZATION

MUMBLE (message)

argument: a message
return value: none

'The initial tree:



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)
```

Start the Controller at
Dispatch on current-contents

The Controller

Process-node (current-node)

argument: a node
return value: none

↓

Enter-node: (Foreach feature of current-node
do (evaluate (get-grammar-routine 'enter-node feature)))

↓

(Foreach slot in (constituents current-node)
do (process-slot slot))

↓

Leave-node: (Foreach feature of current-node
do (evaluate (get-grammar-routine 'leave-node feature)))

↓

RETURN

Process-slot (current-slot)

argument: a slot
return value: none

↓

current-contents <= (contents current-slot)

↓

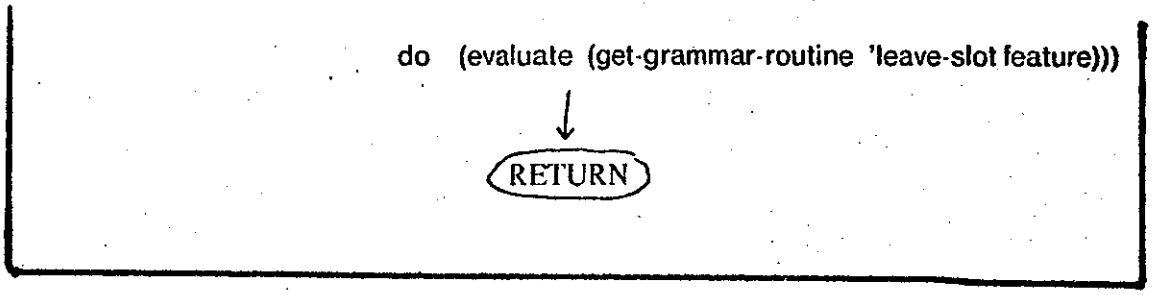
Enter-slot: (Foreach feature of current-slot
do (evaluate (get-grammar-routine 'enter-slot feature)))

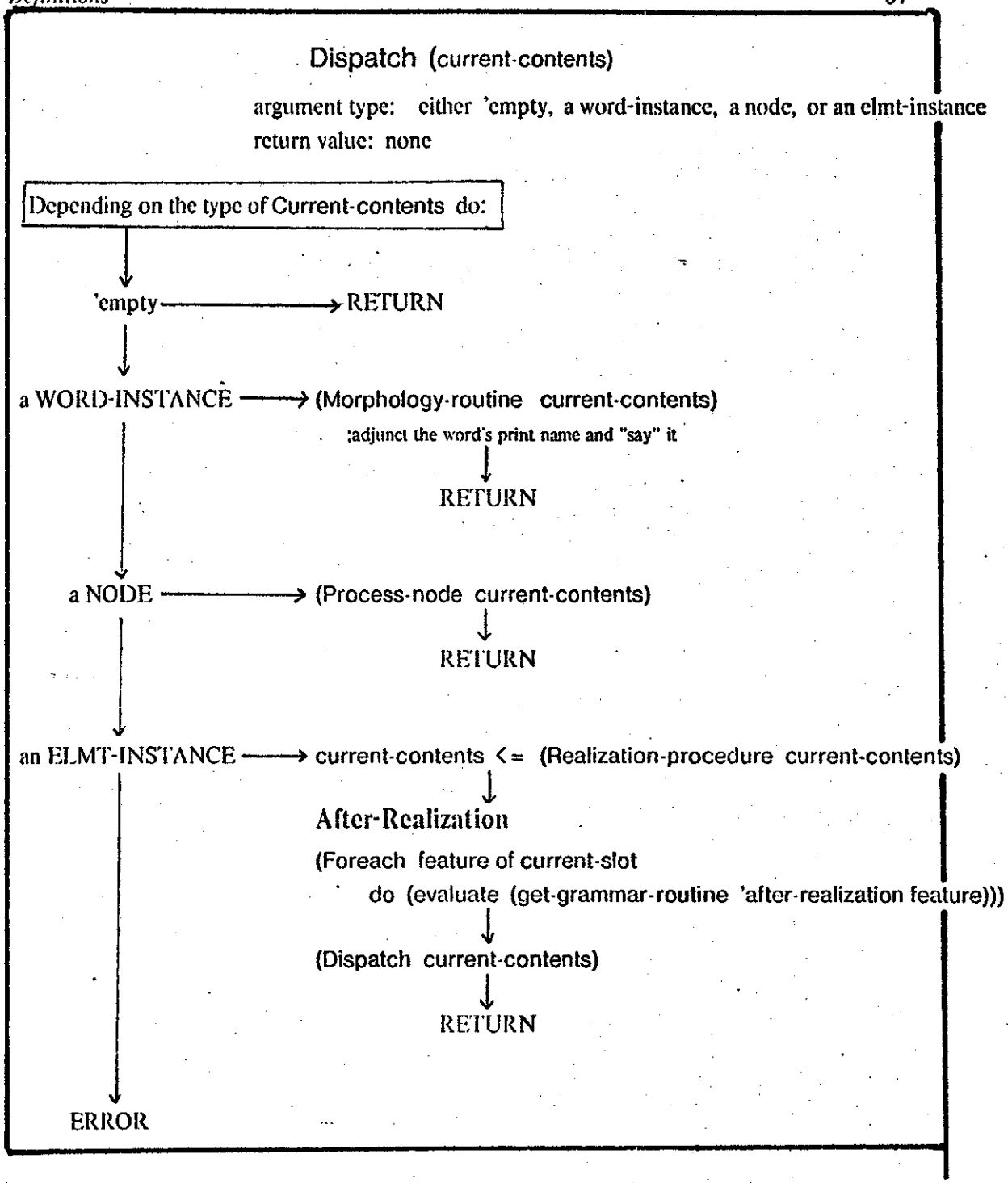
↓

(Dispatch current-contents) (A)

↓

Leave-slot: (Foreach feature of current-slot


do (evaluate (get-grammar-routine 'leave-slot feature)))
RETURN



Initialization

When a new message is passed to the linguistic component, it becomes the contents of the slot "root-constituent" If, at that time, the component has finished realizing the earlier messages, then the current position of the controller will have returned to the top of the tree and the

realization of the new message is begun immediately. If, on the other hand, the controller is still positioned in an earlier part of the tree, then nothing happens to the new message immediately: the controller continues to traverse the tree and to bring about the realization of the earlier message(s) without disruption until the top of the tree is finally reached. (This technique of reusing the same slot is only feasible if the speaker and the linguistic component are operating online, i.e. if the speaker will not accumulate messages faster than the linguistic component can realize them.)

The Tree-traversal Algorithm

The core of the controller is a tree-traversal algorithm. It defines the order in which the parts of the tree will be visited, and directly controls the two basic operations of the component: realizing embedded message elements and printing out the words of the text as they are reached. Interwoven within the traversal algorithm are the data-directed dispatch system that controls the active aspects of the grammar, and the system of grammar-variables that maintain the deitic aspects of the linguistic context.

The tree-traversal algorithm itself is unusual since at the same time that the controller is traversing the tree, it is extending it: replacing message element leaves with more constituent structure. The "extension step" of the algorithm is the recursive call to dispatch made in the elmt-instance case of that same procedure. In effect, the traversal process pauses at each embedded msg-elmt while the realization procedure is run to compute the portion of the tree on which the traversal is to continue. This property of the algorithm is important because it is the basis of progressive refinement and consequently central to the design. At the same time, it raises the algorithmic question of guaranteeing termination: can we be sure that the extension will ever stop?

The extension stops when the phrases being selected by the realization procedure no longer contain embedded ELMT-INSTANCEs. This makes termination the responsibility of the message enumeration function, i.e. if the number of elements in a given message is finite and their enumeration acyclic, then the realization procedure will eventually exhaust them and the expansion will terminate. Notice that this is a speaker-related criterion, not a linguistic one. If the speaker could somehow arrange to transfer an "infinite" message to the linguistic component, the component would not complain and would try valiantly to complete a realization of the message. In fact, if the text resulting from the message were right or left recursive, the component could not even tell that anything was out of the ordinary. Center embedded text, however, could raise stylistic flags pg.(<center_embedding>).

2.4 Attached procedures

The controller's travel through the tree may be naturally decomposed into a sequence of generic events defined in terms of specific points in the tree traversal algorithm. In this theory, these generic events play the same role as "making an assertion" or "deleting an assertion" do in PLANNER-style data-bases, namely they are points for *procedural attachment.* In the linguistic component, attached procedures are used exclusively by the active aspects of the grammar and the procedures are thus known as grammar-routines.

Grammar-routines are attached to individual grammatical categorys, constituent slot-names, and the features associated with them (see section grammar_routines_n.y.w.), and are specific to individual controller events chosen from the following "generic" events:[2] "*entering a* NODE", "*entering a* SLOT", "*after realizing a* MSG-ELMT", "*leaving a* SLOT", and "*leaving a* NODE". These associations will be clear in the way grammar-routines will be referred to in this text, namely as the combination of grammatical label and generic event enclosed in square brackets, e.g. [clause enter-node], [c1 leave-slot], etc..

Generally speaking, the execution of a GRAMMAR-ROUTINE can cause any or all of the following classes of actions. This is discussed in more detail in section grammar_routines_n.y.w..

(1) The immediate print-out of a function word, without the word ever having occupied a slot.

(2) A GRAMMAR-VARIABLE (see below) to be set to some value that the routine computes.

(3) Refining the current-contents by adding hooks to nodes or attachments to elmt-instances. "Lowering" the current-contents into a slot of a new phrase that the grammar routine builds.

(4) The annotation of some part of the tree or some part of the discourse history.

Non-specific GRAMMAR-ROUTINES

The bulk of the grammar-routine s in MUMBLE's grammar are associated with specific constituent-structure-label. However, there are grammatical phenomena whose implementation calls for a routine that will run on every occasion of a given controller event, not just those where a particular feature is present. One such is the definition of a sentence, on pg.<sentences_n.y.w.>. These routines (there can at most be five of them, MUMBLE uses two) are indicated as [<event> *always*].

---

2. In this vein, the HOOKs for extra-constituent information (sect. II.4.5) can be seen as the attachment points for default-decisions&generic events inside the realization-procedure.

Controller variables

The controller maintains three variables: current-node, current-slot, and current-contents, that may be referenced by the grammar-routines. As the flowchart shows, these variables are set and reset recursively as the controller calls its subroutines.

Grammar variables

Without exception, all actions performed by the linguistic component are performed with respect to the current position of the controller within the tree. This is inevitable given the data-directed design since all dispatches: to the morphology routine, to GRAMMAR-ROUTINES, or to the realization procedure, are performed by the controller. This is the single most compelling fact for the design of the grammar since it severly limits what can be a used for the representation of context. Grammatical facts and generalizations must be represented in such a way that the objects to which they apply can always be determined even though the point of view from which these objects would be described is constantly changing.

The key to this deictic representation of context is a set of GRAMMAR-VARIABLES, whose values range over objects in the tree. They are set, reset, and remembered during recursions by explicit commands that are part of grammar-routines. What variables there are and how they should be maintained is a question of how one chooses to do the grammatical analysis, and will be taken up extensively in chapter three. The broad philosophy of how they should be used is discussed later in this chapter in section grammar_variables_n.y.w..

3. Conventions in the grammar

The linguistic component deals with information in three different representational levels: the message, the surface level syntactic structure of the tree, and the pnames of WORDs. At each of these levels, certain relations must hold if the output texts are to be legitimate English. These relations are described by the grammar.[3] The goal of the rest of this chapter is to review the representational devices that make up a grammar for the linguistic component, and to explain the conventions that govern the assignment of particular devices to particular phenomena and their motivation by the particular exigencies of language production.

3. N.b. message-level relations are not considered part of the grammar of the linguistic component even though the well-formedness condition on messages or any constraints that might be formulated in an interlingua are critical to the component's design. This is because messages are constructed by the speaker and it is therefore the speaker that is responsible for representing and obeying any "message grammar".

The grammatical conventions for surface structure will occupy the bulk of the discussion. Surface structure is the level that carrys the greatest burden, being used both to determine the execution sequence of the component as a whole and to represent the current intentional and grammatical context for decision-making. At the end, the morphology routine will be described. In MUMBLE's text-based implementation, this routine is very simple; however, if actual speech were being produced, much more elaborate processes would be required: in particular, a very close interface to the syntactic structure would be required if intonation contours were to be produced incrementally as texts currently are.

This section begins by discussing what a grammar is and what it is to account for in the production process. The point will be made that the meaning of a grammatical term is given by the expressions that refer to it and, ultimately, by the actions that happen when these expressions are interpreted. Following in this line, we next will look closely at why an *explicit* syntactic structure is so important in this linguistic component, and then, expanding on the principles developed there, The specific representational devices of the grammar and the conventions governing their use are the subject of the section following this one.

3.1 What is a Grammar?

In this theory, a *grammar* is a set of objects of certain types.[4] Nothing else need be specified since the behavior and possible interactions of all of the objects follow from the already established definitions of their generic types.

Of course, merely defining a set of objects does not automatically make it a grammar of the English language. The content of a descriptive grammar of English such as [onions], [quirk_&_greenbaum] or [fowler] is not in its choice of terms, e.g. *subject, predicate nominative, participle*, etc., but in the statements that relate those terms: "a predicate nominative agrees in number with its subject", "the implied subject of a participle used as a clause adjunct is the subject of the clause to which it is adjoined". So it is when a grammar is used to constrain production. Here, the equivalents of the descriptive statements are the fixed control and realization procedures and the patterns of PREDICATEs and ACCESS-FUNCTIONs in the DECISIONs and GRAMMAR-ROUTINEs. The properties of the objects in the grammar must be carefully chosen so that the relations that tie them together are appropriate to English.

Adequacy

4. Every object in the linguistic component is a member of one of three possible "realms": the dictionary, the grammar, or the context. The data-types of objects which are part of the grammar are: CATEGORY, CONSTITUENT-SCHEMATA, SLOT-NAME CATEGORY-FEATURE, SLOT-FEATURE, REGION-FEATURE, HOOK, GRAMMAR-ROUTINE, DEFAULT-DECISION, grammar-decision, CHOICE, PHRASE, MAP, SLOT-PATH, MARKING-ACTION, TRANSFORMATION

Obviously the grammar must be so designed that the texts the linguistic component produces are grammatical[5] English: verbs must agree with subjects and not objects, predicates must follow subjects and not visa versa, and so on. This *descriptive adequacy* is a requirement that all language systems: production systems, comprehension systems, and "competence" systems (such as the many types of transformational generative grammar) must satisfy.

However, when we come to the "invisible" underlying linguistic relations of the text such as the choice of categories or the precise patterns of the constituent structure, it is not so clear what relations the grammar is supposed to enforce. Is there a verb phrase? Is the constituent structure of the auxiliary flat or a right-recursive tree? Questions such as these are part of long-standing disputes in the linguistics literature. Their answers are predicated on differing meta-theoretical decisions which assign different explanatory roles to the same representational devices. One cannot determine whether or not the underlying grammatical representations posited by a given grammar are adequate for English without first determining the theoretical framework according to which the grammar was designed. Establishing this framework is thus first order of business:

3.2 Why use constituent structure at all?

Practically all goal-directed language production programs prior to the present one did not bother to build an explicit constituent structure for the texts they produced.[6] One must then ask why constituent structure is so important in this design. This is really two questions: (1) why must any structure be built at all?—why can't the words be produced directly? and (2) if one should be built, why an essentially classic grammatical constituent structure?

3.3 Decisions as plans

Any natural language producer, human or machine, is only capable of saying one word at a time; yet, nearly every decision that is made in the course of production entails the simultaneous specification of temporal and grammatical relations involving more than one word (or more abstract units). It follows that there must be some provision in the system for *remembering* what has been decided until such time as all the actions required to realize those relations have been carried out (i.e. until all of the words involved have been given the right morphological markings and have been printed). Unless some record is kept, the system will either forget what it has planned to do and stop talking, or it will be continually "re-making" its decisions, with the possibility of inconsistency.[7]

---

5. This statement requires a caveat. If the process in which the grammar operates is not responsible for certain aspects of the final text—such as its length—then there is no way that it can, or should, be able to insure the "grammaticality" of those aspects. In particular, I will argue later (ross_constraints_n.y.w.) that the component's grammar should not be responsible for Ross's island constraints because they are best realized at the level where messages are constructed.
6. To the best of my knowledge, the only exceptions as of August 1979 have been [gretchen_german] and in some respects [Davey]. See [ddm_past].
7. The production model of [kempen_1979] is such a case. His notion of "piecemeal sentence production" calls for the continual regeneration of the earlier parts of each sentence as successive conceptual elements are added to it. See also [henry_thompson].

In this context, a "decision" is always a decision to perform some action(s): utter a word, create a grammatical relation, realize a subelement, and so on. (N.b. one can especially decide to schedule further decisions!) In effect, a decision is a commitment to execute a certain (minimally specified) procedure at a certain time. In a data-directed system, this commitment is made by building the requisite data structure (e.g. a NODE) and positioning it in the path of the controller at the appropriate point with respect to all the other things one has planned to do.

ATN-based production programs (e.g. [slochum][goldman][wong][shapiro_acl79]) avoid constructing an record by using an ATN in which all of the alternative execution path segments that could be selected have already been multiplied through and are part of the explicit state space of the transition network The advantages of the data-directed technique are: (1) the record is vastly easier to inspect; (2) being declarative, it can be easily changed if necessary (e.g. for in-line heavy phrase shift); and (3) as they are usually designed, an ATN that was both to be indelible and tobe controlled by the contents of the message (rather than using the standard, grammar-based search) would have an incredibly large state space, so much so as to make its design impractical.[8]

3.4 Incremental realization

Even if some structure must be built to record decisions, why must it be a grammatical one? Earlier production programs such as [chester] or [shrdlu] kept their records in the form of nested lists of calls to production functions—what is the purpose of a grammatically annotated record? The most obvious utility of a grammatically annotated record is that it can be inspected by decision making routines, thereby allowing the grammatical context to directly influence the decision making. However, this is actually only a corollary of the more important reason which involves how one implements a policy of incremental realization in a data-directed system.

The cornerstone of incremental realization is Marr's "Principle of Least Commitment" [marr_least_commitment] (rephrased for production rather than analysis).

> *No decision-maker should be forced to specify more detail in its decisions than can actually be justified on the basis of the readily available information.*

This principle has two kinds of force here. First of all it is the basis of constraints in the theory governing what knowledge resources an ENTRY should need to appeal to, i.e. (1) no ENTRY should be forced to know any of the details of English grammar (this is the point of having a *linguistic* component); and (2a) ENTRYs should not need to know anything about the structure of subelements that have their own ENTRYs; (2b) linguistic generalizations over subelements (e.g. TRANSFORMATIONs) should treat them as atomic objects.

---

8. By "usual design" I have in mind equating MUMBLE's dictionary entries with ATN subnetworks (as though they were np's or clauses in an ATN parsing grammar) with the decision-rules becoming the tests on the arcs and the phrase-schema becoming subnetworks that the atn would traverse.

The second force of the principle is as a call for modularity: no decision-maker should make a decision that one of its subordinates would have to make anyway in other circumstances. Instead of making the "redundant" subordinate decision itself, each ENTRY will include a token(s) in the record of its decisions that will trigger a subordinate decision-maker at a later point. Modularity in the design of the decision-makers reduces the amount of redundant decision-making knowledge in the system and reduces the number of dependencies on the individual decisions because each will require less information to function.

As a concrete example, consider the problem of recording a decision to select a simple subject-predicate construction. (e.g. deciding to map man(x) and mortal(x) into *All men - are mortal*) In making this decision, an ENTRY should not have to simultaniously mark the number of the verb; instead, the fact that this action is needed should be part of the record. This is because the verb cannot be marked before the number of the subject is known, a datum which, at the time when the decision is made, could be learned only by having the ENTRY look ahead to see how the subelement that is targeted for the subject position will be realized. Such look ahead is a violation of the incremental realization constraint because it would require the ENTRY to incorporate extensive, redundant information about the internal structure of that element (i.e. enough to determine its number—sometimes effectively its whole ENTRY). Why should an ENTRY have to go to this effort if the marking of the verb can be postponed until after the subject has been realized and its number is trivially available?

However, if the verb-marking action is to be postponed, the record of a "clause_subject-predicate" decision will have to be specially labeled so that when the time comes the controller will know to initiate the postponed action. Just recording the two elements as a list (essentially what is done in systems based on nested function calls) is not sufficient; the annotation on the "list" must at least be rich enough to identify which element will carry the number marking and which element it is to agree with.

Of course, the labeling does not have to be as specific as *do-number-agreement.* All that is required is that the chosen label(s) *imply* the specific label through some system of intermediate actions and conventions. If the decision-recording language is sufficiently rich, an ENTRY can specify entire systems of later decisions without having to know what they are individually. This, of course, is the point of a grammar. Grammatical terms like *clause* or *noun phrase* stand for extensive sets of actions and contingent decisions which ENTRYs do not have to know anything about. This function of grammars has long recognized and emphasized by researchers in the *systemic grammar* paradigm. [Halliday_big_paper][Halliday_modality_mood][shrdlu] [Huddleston_generation][davey].

A text in the English language must obey the constraints of English grammar. It follows that the relations that linguists have selected for describing those constraints should be directly

applicable to production.[9] A data-directed control structure allows the designer of a production system to take advantage of linguists' results directly by using the very relations that the linguists have decided upon as part of the decision-making record.

At the same time, the design decision to employ a classic grammatical constituent structure as the record was not made for its own sake. MUMBLE's constituent structure is less extensively annotated than many of those in the linguistics literature and will often differ from them in significant ways (see below). This is because the metatheoretical role of constituent structure here, i.e. to record decisions, is not the same as it is in, e.g., the Aspects model of transformational grammar [aspects]. The grammatical facts that must be accounted for are the same and therefore the notations have a great deal in common; however, the way in which the linguistic component uses them is particular to production.

3.5 Constituent structure design for production

3.6 Message structure

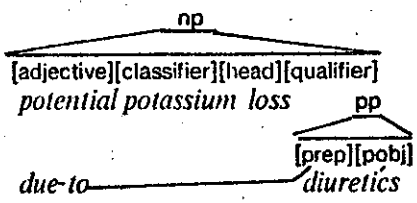
The most important function of the constituent structure is to be the repository of yet to be realized message elements. This means that the conceptual structure of the messages—how they decompose into message elements—constrains the shape of constituent structure in a very specific way. Message elements (or more precisely ELMT-INSTANCEs) will occupy SLOTs in the tree—one message element, one slot—therefore, when an ENTRY decomposes a given message element into, say, three subelements, it must select a PHRASE with three free SLOTs to put them in.

When there is a conflict between the decomposition pattern of a message and the pattern of nodes in the usual constituent analysis of the English construction selected for it, the requirements of the message structure take precedence over the grammatical analysis and a different technique is used to achieve the grammatical effects.

A case in point is the treatment of noun phrases for the Digitalis Advisor. A phrase like: "*potential potassium loss due to diuretics*" would be analysed in a systemic grammar (for example [shrdlu] pg.56]) with an almost "flat" constituent structure, i.e.

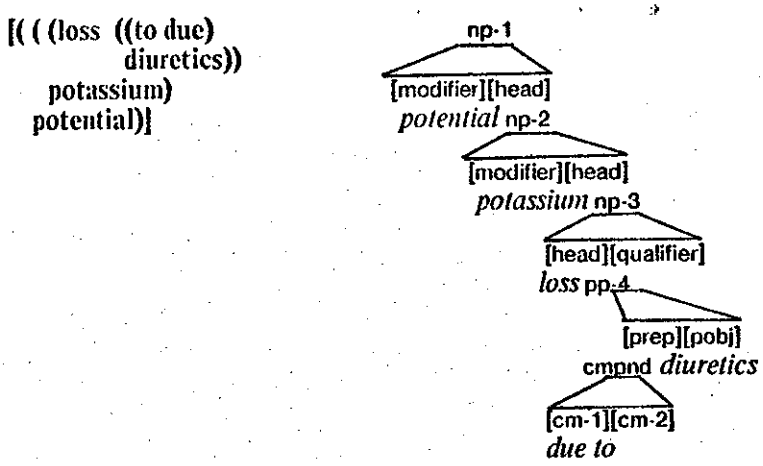
---

9. This is provided, of course, that it is possible to interpret the terminology of descriptive grammars of English in terms of potential decisions or actions in the language production process. Certain modern transformational-generative representational devices, such as "over-generate and filter" [chomsky_&_lasnik], cannot be accommodated without severely violating the indelibility constraint.



But the OWL expression which engenders that noun phrase is a deep binary tree. Each node of that tree is an OWL "concept", either atomic or comprised of a "generalizer" and a "specializer", each of which is itself a concept. The natural way to realize such a structure is one concept at a time, embedding the generalizer and specializer at each level into their own constituent structure SLOTs.

This means the final constituent structure will have one syntactic NODE for each OWL concept, as shown in the figure.



Any potential aesthetic "strain" on the set of "normal" syntactic categories is offset by the simplicity of realization process that is gained thereby.

3.7 An active grammar

Earlier sections of this chapter described several representational devices: GRAMMATICAL-DECISIONS, DEFAULT-DECISIONS, and GRAMMAR-ROUTINES, that have the power to evaluate expressions that test the current context and perform contingent actions that may print texts directly or modify the tree. These devices are quite unlike ordinary passive devices such as CATEGORYs or SLOT-FEATUREs which can only "act" indirectly by influencing the outcome of DECISIONs that reference them. One may ask why active devices are needed—why wouldn't it be

possible to avoid later active elaborations to the constituent structure by building a more elaborate structure in the first place? The reason is that it is typically *impossible* to build more elaborate structures at decision-time because their underpinnings—the sentence-initial words to be capitalized, the pronouns to be marked for case, the head nouns to be marked plural, etc.—do not yet exist and cannot exist because they depend upon decisions that have yet to be made. This is, of course, a consequence of the incremental realization constraint. Active devices in the grammar are the price one pays for being able to avoid redundant lookahead and anticipation at decision-making time.

Grammatical actions can only take place when triggered by events in the controller (<events_in_the_controller>). If the first word of each sentence is to be capitalized, then there must be an event that can be associated with the beginnings of sentences; if there is to be a comma after every item in a list, then there must be some way to mark lists as such and to note the end of every item in them. This aspect of the design becomes important whenever decisions are to be made whose defining conditions are strictly linguistic. Two such decisions are "particle movement" (saying *put down the block* versus *put the block down*) and "extraposition" (*Persuading Macbeth was easy* versus *It was easy to persuade Macbeth*). Both of these sets of alternate orderings are only relevant for certain prior choices of main verb, consequently their triggering events must take place after the verb has been selected but before the phrases are begun.

In general, we would like decisions like particle movement or extraposition to be an automatic part of the grammar—something that individual ENTRYs do not have to be concerned with. To make this possible, we must make certain that the ENTRYs choose PHRASEs whose structure (i.e. choice of CATEGORY-FEATUREs, choice of SLOT-NAMEs, etc.) insures that the triggering events for these decisions will take place. This aspect of the design is the one most responsible for fixing the details of the grammar.

3.8 The 2-color hypothesis

Once the principle of an active linguistic representation has been established, it is a short step to the 2-color hypothesis:

> *The only objects that may legitimately occupy* SLOTs *in constituent structure are* ELMT-INSTANCEs *or* WORD-INSTANCEs *explicitly chosen by a dictionary* ENTRY.

This is a psycholinguistic hypothesis rather than a computational one and has been incorporated into this theory as a *discipline* to guide otherwise open design choices about which closed class words should appear directly in the tree and which should be an implicit part of the constituent structure.

This hypothesis appears in the linguistic component as a constraint on the designers of the grammar and dictionaries forcing them to insure that constituent structure slots are occupied only by objects specifically selected by some ENTRY. In operational terms, that means that the only

way into a constituent slot is as an argument in a CHOICE-APPLICATION. No WORDs or morphemes selected by the grammar are to occupy SLOTs at any time; they are all to be printed directly by GRAMMAR-ROUTINEs. Closed class words and morphemes become, in effect, a direct extension of the syntactic structure: they may not be passed through the morphology routine or further modified by grammatical context in any way, and there will not be any independent record of their presence in the text—only what can be deduced from the record of what CHOICEs were made.

The hypothesis is prompted by the differing distributional patterns of open and closed class words in speech errors, particularly full word exchanges: *What kind of hungry are you going to be food for?*, and errors that "strand" grammatical morphemes: *...you can see my friend checking cashes.* Error data like these (taken from my own collection) lend themselves to an interpretation as shifts of salient words relative to a fixed phrasal matrix. (See [garrett_latest][Klatt_&_errors] [ddm_squibs] for elaboration.) The simplest explanation of any error is that it is caused by the malfunction of some part of the *normal* processing mechanism. Consequently, if open-class words and grammatical matrixes are as differentially effected by speech errors as seems to be the case, then we will hypothesize that, in normal processing, the two are represented differently and manipulated by different operations. Metaphorically speaking, they are painted two different colors.

It is not obvious *a priori* that this hypothesis can be incorporated into the linguistic component in a way that is consistent with other design goals such incremental realization or the direct use of data structures from the expert program. But if, in fact, a coherent analysis is possible (as appears provisionally to be the case with MUMBLE) then we will have computational evidence for the sufficiency of the 2-color hypothesis as a psycholinguistic model.

3.9 The meaning of grammatical objects

The objects used to label constituent structure: CATEGORYs, SLOT-NAMEs, CATEGORY-FEATUREs, SLOT-FEATUREs, and HOOKs, all take on meanings in two different ways: as symbols and as attachment points for GRAMMAR-ROUTINEs (<attached_procedures>).

3.10 Symbols

In any formal system, the meaning of a symbol derives from the axioms in which it is mentioned [Hayes_defense_of_logic]. Correspondingly, in the linguistic component the meaning of an object *qua* symbol derives from the expressions: DECISIONs and GRAMMAR-ROUTINEs, in which it is mentioned, where being "mentioned" means that the object's name is a literal part of the expression. The *predicate_entry* (pg.209) is a good example. Its decision-rules make reference to several SLOT-NAMEs (*modifier, head, predicate*) and SLOT-FEATUREs (*clausal, nominal*); in so doing, they add to the definition of those labels by specifying CHOICEs which they will trigger, and

(implicitly) that they are a disjoint, discriminatory set.

In any system, the most "interesting" symbols are those that control entire systems of choices, the major grammatical categories such as *clause* or *noun phrase* being well known examples. However, there are many, more subtle examples. One such is MUMBLE's REGION-FEATURE *formal*, originally developed for the logic domain. As its name suggests, it is intended to influence the style of the text—that vague dimension on which breakfast conversations and legal contracts fall at opposite poles. Its meaning in MUMBLE however is not at all vague, though it is certainly incomplete.

Formal presently controls two effects: the use of contractions and the realization of the logical connective IFF. The contraction of *not* to auxiliary verbs or of modals to subjects occurs or does not occur depending on the CHOICE of a GRAMMATICAL-DECISION made within the morphology routine. (Each contraction opportunity entails a new decision.) At the moment, contractions will go through unless *formal* is present, i.e. the decision used is:

```
(define-decision use_a_contraction?
    gating-condition (not formal)
    default (use_contraction))
```

Similarly, the ENTRY *iff_entry* (212) includes a CHOICE-FILTER which, when (not formal) is true, eliminates *realize_consequent_as_restrictive-relative* as a possible CHOICE.

Eventually, one would expect phenomena such as the use of *who* versus *whom*, or of dangling prepositions, or even certain word choices, to come within the system controlled by *formal*. Whether this would in fact be a correct analysis depends on whether these phenomena form a coherent set and should be controlled by a single atomic decision, i.e. is it always that case that all of these phenomena are appropriate when any of them are? Consider: the dictionary for the logic domain only assigns the feature *formal* to regions dominated by SLOTs with the feature *conclusion* (in the sense of "the conclusion of an argument" see pg.<ui_argument_n.y.w.>) In so doing the dictionary has associated the intentional description *conclusion* with a set of particular linguistic behaviors. If it should turn out that future "conclusion" texts produced by this dictionary do not have the intended effect on their readers, then we will then have evidence (perhaps the only possible kind of evidence) that the system for *formal* must be redesigned.

3.11 Procedural attachment points and Data-types

Data-types are analogous to names in the way they acquire meanings. In effect, they are second order symbols since, by using operations that make reference to specific data-types, the definitions of the fixed procedures create schematic expressions that dictate the meaning that objects take on by virtue of their data-type. Probably the most important part of this "meaning"

is the difference between objects of different data-types as places to attach GRAMMAR-ROUTINES. CATEGORYs and SLOT-NAMEs are parts of differently placed events in the controller and as a consequence, their attached procedures will differ in the parts of the tree they may effect; the kinds of information their PREDICATEs will have ready access to; the timing of their actions compared to others'; and other questions of that sort.

[Enter-slot] and [after-realization] are a good example. The figure on page <attached_procedures_figure> shows that, from the point of view of the final text, these two attachment points are assigned to contiguous events, i.e. they take place after all of the text dominated by the previous SLOT has been printed and before any printing for the current SLOT has begun. As far as the final text is concerned, they are equivalent sources for function words; however, from the point of view of what kinds of information are readily available when such routines run, they are quite different.

To put some flesh on this example, let us consider the particular case of the SLOT-NAME *complement*. Verbs like *expect* and *persuade* take "sentential complements"—message elements that would become regular clauses if they were not embedded. Depending on whether the subject of the embedded clause is (would have been) the same as subject of the main verb and on whether modal or tense are used, these complements will appear as infinitive verb phrases introduced by *to* or as clauses (optionally) introduced by *that*:

"Lady Macbeth expected Macbeth to become king."
"Lady Macbeth expected that Macbeth would soon be king."

In MUMBLE, such message elements appear in [complement] slots (but see II.4.7). Consequently, the function words *to* and *that* must be produced by GRAMMAR-ROUTINEs attached to the SLOT-NAME *complement*. The question is, should they be attached at [complement enter-slot] or at [complement after-realization]?

The choice of *to* versus *that* depends upon whether message element in [complement] is realized as a verb phrase or as a clause. If the attachment were made at [complement enter-slot], this realization would not yet have happened (cf. diagram on pg.<attached_procedures_within_controller>). Conceivably, the GRAMMAR-ROUTINE could know how to look ahead and deduce what the realization would be (see, for example, section <analysis_routines>). On the other hand, if we were just to wait and attach the routine to [complement after-realization], the needed information would be trivially available and the routine can be very simple, i.e.

```
(define-label  to-that
 after-realization
    (lambda (contents)
        (cond ((eq (category-of contents) 'clause)
                (make-and-execute-decision  'whether-to-say_that_before-complement))
               ((eq (category-of contents) 'vp)
                (mprint '|to|)) )))
```

4. Representational Devices for the Grammar

By the 2-color hypothesis (pg.<2_color_hypothesis>), whenever an ENTRY makes a (matrix) CHOICE, it is selecting a parameterized structure, L [e_1, e_2, ...], where "L" is an extended surface structure and the subscripted "e"'s are message elements. *Extended surface structure* is the term I will use in the rest of this paper to describe the linguistic level of representation given in the tree. The form of the tree specifies all the linguistic relations among its constituents: syntactic, discourse, and intentional, all the function words and grammatical morphemes that will appear in the text, and all grammatical constraints on the realization of any ELMT-INSTANCEs embedded within it.

This section will first show how the different kinds of constituent-structure-labels are defined and what they may be used for. It then looks at GRAMMAR-ROUTINEs and how they are used to extend the usual notion of surface structure. Finally, certain fine points in the specification of constituent structures will be considered—the different ways of specifying an extended surface structure will be shown to correspond to the different times at which decisions are made in the realization of a message and what decision-recording structures can be expected to exist then.

4.1 CONSTITUENT-SCHEMATA

CATEGORYs and SLOT-NAMEs, the generic grammatical objects corresponding to the NODEs and SLOTs of constituent structure, are defined in terms of CONSTITUENT-SCHEMAS. These, you will recall, are the representational device used by PHRASEs to specify immediate constituent structure; they are thus the means (the *only* means) whereby particular CATEGORYs or SLOT-NAMEs become part of the tree. CONSTITUENT-SCHEMA have the following properties.

name A CATEGORY-FEATURE, automatically part of the features of any NODE created by instantiating the schema.

features A list of zero or more CATEGORY-FEATURES or REGION-FEATURES which also will become part of the features of any NODEs instantiated from the schema.

category A CATEGORY.

slots A list of one or more SLOT-NAMES.

In MUMBLE, CONSTITUENT-SCHEMAS are specified with the load-time operator *define-schema.*

```
(define-schema vp_predicate-nominative (vp)
    slots (mvb pred-nom))

(define-schema regular-np (np)
    slots (det modifiers head qualifiers))

(define-schema regular-pp (pp)
    slots (prep prep-obj))
```

Instantiating a CONSTITUENT-SCHEMA NODEs are created from CONSTITUENT-SCHEMAS by a simple process. First, a new object of type NODE is created. (Note, This is the *only* way that NODEs come into existance.) By convention in MUMBLE, its name is created by appending the next available number (from the sequence used for this purpose) to the name of the category of the CONSTITUENT-SCHEMA ("schema" for short). The features property of the new NODE is a newly created list consisting of the schema's category, its name, and its features if there are any, in that order.

Then the new NODE's immediate-constituents property is set. This is done by taking each SLOT-NAME on the schema's slots property and creating a new object of type SLOT with that SLOT-NAME as the value of its slot-name property and a empty contents property (i.e. its value is the SYMBOL *nil*). These SLOTs are accumulated in a list, maintaining the order given by the schema's slots property, and the list, when complete, is installed as the NODE's immediate-constituents.

Newly created NODEs have empty SLOTs, and no hooks or record properties. All of these are supplied by PHRASES (or by subsequent TRANSFORMATIONS or GRAMMAR-ROUTINES). Instantiating a PHRASE consists of instantiating its first CONSTITUENT-SCHEMA, then filling each of the indicated SLOTs applying the process recursively when the indicated contents are another CONSTITUENT-SCHEMA. The features property of each created SLOT is copied from that of its slot-name plus any features specified locally with the PHRASE.

Schema as constituent structure labels The name of a CONSTITUENT-SCHEMA can be used directly as a symbol of the pattern of SLOT-NAMEs it denotes. So for example, if a program knows that some constituent is a *regular-np,* its does not have to explicitly scan its immediate-constituents

in order to learn that it has a [determiner]. That fact is implicit in the name of the CONSTITUENT-SCHEMA.

CONSTITUENT-SCHEMAS are also the logical place to attach any GRAMMAR-ROUTINES that may be necessary to signal decisions about this pattern of constituents that must be postponed until after the schema has been instantiated. This is why the schema's name is automatically a feature of any NODES created from it. For example, the default main verb of a predicate nominative construction is always the copula *be*. This fact is represented by the [pred-nom after-realization] GRAMMAR-ROUTINE:

```
(define-label  vp_predicate-nominative
 after-realization
    (lambda (the-node)
        (cond ((not (getslot 'mvb the-node)) ;the slot is empty
               (fillslot  'be 'mvb the-node))) ))
```

Similarly, when "derived" message elements are created through the merger of almost identical relations at the message level (merging_message_level_relations_n.y.w.), their number properties will not be known untill the dust settles (as it were). Texts like,

"*Pp has the roles: pobj, prep, interp, and ppobj.*"

are built by this grammar:

```
(define-schema  np-name  (np)
 slots  (det head name))

(define-label  np-name
 enter-node
    (lambda (the-node)
        (cond ((and (nodep (getslot 'name the-node))
                    (eq (category the-node) 'conjunct)))
                ;if there are several "names"
                ;they will have been seen earlier and
                ;made into a constituent.
              (set-hook the-node 'number 'plural))
              ))
```

Constraints on CONSTITUENT-SCHEMA

Each slots property may have only one instance of a given SLOT-NAME. This restriction is necessary to insure that SLOT-NAMES define *unique* positions in a NODE's immediate constituent structure. The SLOT-PATHS used in CHOICES to define the mapping between their parameters and positions in constituent structure are given entirely in terms of SLOT-NAMES; thus this restriction is

necessary to insure that the PHRASE instantiation operation can be consistantly defined.[10] (Any cases where duplicating a SLOT-NAME within one schema might seem to be useful as a way to encode some common property are better handled by assigning the property to SLOT-FEATURE which can then be assigned to all of the individual SLOT-NAMES.)

It is impossible in this theory to introduce a CATEGORY or a SLOT-NAME into the grammar except via a CONSTITUENT-SCHEMA that relates them.[11] One cannot define a CATEGORY without simultaniously specifying at least one set of SLOT-NAMES for its immediate constituents. Similary, one cannot define a SLOT-NAME without simultaniously specifying its temporal order with respect to the SLOT-NAMES of at least one other constituent-schema.

Schema with arbitrary numbers of constituents Conjunctions and simple discourses are atypical grammatical categories in that they may have any number of constituents, all of which have identical properties (except possibly the first and the last). These unorthodox CATEGORYs call for unorthodox CONSTITUENT-SCHEMAS. In MUMBLE, they are defined in terms of a "kernal-slot-name") (e.g. "c" for conjunctions), from which individual SLOT-NAMES are created by appending a number; new SLOT-NAMES will always be created in the same order (i.e. [c1], [c2], ...). To assemble a, e.g., conjunction NODE from a list of (proto-) constituents, each constituent is taken in turn, paired with the next-slot-name in a newly created SLOT, and the SLOTs accumulated in the NODE's immediate-constituents property.

The operation of computing the next slot-name of a node with one of these schema can be done incrementally while the process is running. That is a function has been defined, call it next-slot-name, which is given a node as input, updates its next-slot-name, construction a slot to which it gives that slot-name, and adds it to the node's constituents in the appropriate place. Defined in this way, new constituents can be added to arbitrary-length CONSTITUENT-SCHEMA at any time. The constituents of *root-node* are an important case in point. New messages are introduced into the lingustic component by putting them into "the next available SLOT for a message", which is computed as needed by this function.

Schema as productions CONSTITUENT-SCHEMAS are akin to the productions of a phrase structure grammar ("PSG") in that they specify (albeit indirectly) the set of phrase structure trees permitted by the grammar. That is, where a PSG would use a rule like:

clause -> NP + VP

---

10. Alternatively, we could define multiple instances of the same SLOT-NAME in a CONSTITUENT-SCHEMA to mean that any ELMT-INSTANCE mapped into such a "position" is to be placed in all of the SLOTs with that name. This capability, however, has not proved to be of any utility in MUMBLE.

11. Actually, this is not strictly true from the designer's point of view. The section on "fine points" will discuss the use of TRANSFORMATIONs and GRAMMAR-ROUTINESs to activate "latent" constituent slots, permiting "regular" CONSTITUENT-SCHEMAs to be augmented in predictable ways.

to specify that the first constituent of a clause must be of category NP, MUMBLE would use the comparable CONSTITUENT-SCHEMA:

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

in conjunction with the following CHOICE-FILTER on *subject*:

```
(define-slot  subject
    choice-filters  ((or 'np 'tenseless))  )
```

This filter applies during the realization of any ELMT-INSTANCE contained in a SLOT whose slot-name *subject*, where it is interpreted to permit only CHOICEs whose phrases include either of the category-features *np* or *tenseless*.[12] to be selected.

CATEGORYs and SLOT-NAMES CATEGORYs are labels on NODEs. From the dictionary designer's point of view they can be used for encoding facts or intentions about any subelements that are constituents (immediate or distant) of the same NODE or about the entire region of the tree that the node dominates.

That a msg-elmt)'s subelements will all be grouped under the same CATEGORY follows inevitably from the definition of realization as a substitution operation: all msg-elmts are realized as single-rooted trees (except, of course, those that become WORDs) and the root node *must* have a CATEGORY. Choice of CATEGORY is thus trivially a way of encoding the fact that certain ELMT-INSTANCEs were derived from the same message element. However, to just determine common ancestry it is sufficent to know that the ELMT-INSTANCEs were dominated by a common NODE, regardless of its CATEGORY.[13]

Every NODE in the tree dominates a contiguous section of the final text, and, by the same token, "dominates" the realization decisions that lead to that text. Such *regions* can be described—the decisions within them constrained—by the CATEGORY (and REGION-FEATURES) of the NODEs that define them. The effects of the region will be implemented by GRAMMAR-VARIABLEs or by direct references to the CATEGORY as a symbol. The "ceiling" on a region is, of course, the NODE to which the category is assigned. Its "floor" will be reached either when the controller enters another occurance of the same CATEGORY (since controller variables are

12. In MUMBLE, gerunds, participles, and nominalized clauses are given the CATEGORY-FEATURE *tenseless*.
13. Indeed, in MUMBLE, grouping seems to be important only when it gets in the way, e.g. the case of dynamically adjoined phrases, where dummy NODEs must be created to satisfy the formality that SLOTs may contain only one object and are then "ignored" by the grammar because they have no grammatical properties.

recursive (pg.<controller_variables>)), or when a lower region is entered that is an alternative in the same grammatical system. Subordinate clauses are "floors" for major clause regions, for example.

Typically, this description of a region is linguistic: a realization is labeled as a certain type of syntactic object—a *NP* or a *clause*—and properties of the region covered by the realization follow from that label. Thus from the point of view of the grammar, CATEGORYs have their usual meaning: they are context free descriptions of constituents (NODEs), identifying the properties that they have by warrant of their form rather than because of their position or function relative to other constituents.

The principle context sensitive description of a constituent is given by its SLOT-NAME. The constituent in the [subject] of a *clause*, for example, enters into certain grammatical relations with the constituent in the [predicate] or in any [adjunct]; these relations described in the grammar strictly by reference to those SLOT-NAMEs. (Because SLOT-NAMEs are defined in terms of CONSTITUENT-SCHEMAS, the scope of this description is limited to the region dominated by the covering CATEGORY. The designer may arrange that a SLOT-NAME has a meaning relative to other SLOT-NAMEs not in the same schema, but this must be done by tacit naming conventions in the grammar, and is not governed by the theory.)

SLOT-NAMEs in MUMBLE are the primary representation of a constituent's grammatical function, labeling constituents according to their role with respect to the regions that contain them (e.g. the "objects" of verbs, "determiner" and "head" of NP's, the "object" of a prepositional phrase, and so on). The morphology routine uses this information to determine the case form of pronouns (<determining_pronoun_case_in_the_morphology_routine>); various GRAMMAR-ROUTINEs use it to define number concord; and the thematic TRANSFORMATIONs depend upon it as their basic vocabulary.

SLOT-NAMEs also define regions, although, unlike CATEGORYs, these regions are based on relations relative to sibling regions and not on being part of a containing region. For example, within conjunctions, the GRAMMAR-ROUTINEs assigned to the individual conjunction slot-names maintain the GRAMMAR-VARIABLEs *current-conjunct* and *previous-conjunct*. Another example is in the logic domain, where the top-level discourse structure is based on the sequence of lines in the proofs: each line is contained in an individual SLOT and grammar-variables are defined for the *current-line* and *previous-line*.

Because they have direct effects on the text or on later decisions, SLOT-NAMEs are a natural way to encode any message-level relations among subelements that are to be carried over into the linguistic representation. (This is particularly true when the speaker uses a "context-based" representation (<context_based_representations>).) An interesting example of this kind of encoding appears in the logic domain. The third through fifth sentences of the example on page <barber_proof> (repeated below for convenience) comprise a "subargument"—a pattern of

statements intended to introduce some fact, make an observation, and draw a conclusion.

> *Now, anyone who doesn't shave himself would be shaved by Guiseppi. This would include Guiseppi himself. That is, he would shave himself if and only if he did not shave himself.*

These sentences have a rhetorical structure comparable with the syntactic structure interior to sentences—something that can be represented by a CONSTITUENT-SCHEMA:

```
(define-schema  argument-type1  (discourse)
    slots  (fact statement restatement))
```

This example argument was constructed by taking three relatively neutral logical expressions and embedding them in the context created by that schema, using the CHOICE below. The CONSTITUENT-SCHEMA itself is responsible for the "preambles": "*now,* " (from *subargument*) and "*that is,* " (from *restatement*), and for the inhibition of contraction and the spelled out connective in the third sentence (from the *formal* feature on *conclusion*). The CHOICE augments the context by adding a marked focus (which has kept the second sentence from coming out as "*Guiseppi would be a member of that set*", which would have shifted the focus), casting the entire argument in a modal context, and instigating special effects in the later realization of the first two expressions[14] that lead to the use of *any* and the reflexive *himself.*

```
(define-choice  spellout_universal_reasoning  (univ-formula set-membership-stmt line)
 phrase (subargument-type1  (subargument)
            *hooks* ((focus  univ-formula)
                     (modality  'conditional)) )
 map ((univ-formula . fact)
      (set-membership-stmt . statement)
      (line . (restatement)))
 actions  ((attach-to  univ-formula 'express-as-a-set)
          (attach-to  (substituted-constant  line)
                      ;this returns the constant substituted for the
                      ;universal variable by the "universal instantiation" rule.
                     'intensify  ;the tag
                     nil  ;the value - irrelevent in this case
                     '((eq current-discourse-slot 'statement))
                        ;This argument will fill the TARGET-ELMT property
                        ;of this early-instance.
                     )) )
```

14. This CHOICE was created "by hand". The conceptual base of the speaker in the logic domain is very small, and not capable of deducing on its own that those linguistic devices were the right ones to use. This will be the case for many generations of program speakers to come.

Preambles made up of idiomatic phrases or grammatical function words such as these are exclusively associated with SLOT-NAMEs and SLOT-FEATUREs rather than CATEGORYs.[15] This is for several reasons. Preambles are often used to specialize a general unit for a particular discourse purpose. Since the usual signal that a unit is special is that it is the contents of a special SLOT, this makes the slot-name the natural place to attach any preamble. Also, preambles that might have been attached to a CATEGORY are often easier to express as parts of PHRASEs. (This is the only way preambles can be encoded at all when their texts are sensitive to variable grammatical relations.)

When two or more SLOT-NAMEs or CATEGORYs share some part of their meaning, this should be captured through the use of a *feature.* The attached procedures or direct, symbolic effects that implement the shared effects are then given to the common feature and omitted from the individual objects. If the effects are a permanent of the object's definition, the feature should be included as part of its intrinsic-features, otherwise feature can be specified explicitly as part of a PHRASE.

Examples of such features in MUMBLE are *nominal* and *clausal.* These are used to tell ENTRYs whether they should use noun phrases or clauses for their realization (see, for example, the *predicate-entry* on page 209). *Coordinate* and *subordinate* are used by the pronominalization heuristics (pg.166). The effects of coordination in providing enviroments for elipsis and for parallel CHOICEs are common to many more constructions than just conjoined lists, including "if-then", conjunctions with *but*, and clause adjuncts. These are captured by the SLOT-FEATURE *coordinated-slot.*

The structure-manipulating feature *subject-aux_inversion* (page <subject-aux_inversion>) occurs in other constructions than questions, including "pointing" clauses: *On the left is the living room*, and a literary use of negation: *Never have I seen such exemplary conduct.* The grammatical properties of the various complements, their use of complementizers and the TRANSFORMATIONs they subject to, are given by a system of labels: *subject-to-equi, obj-inf-comp, to_or_that* (section <embedded_clauses>).

Operationally, there is no difference between CATEGORYs and CATEGORY-FEATUREs (or SLOT-NAMEs and SLOT-FEATUREs) in terms of the options for encoding grammatical or intentional facts.[16] Both have the same possibilties for attached procedures and both may be directly tested for their presence or absence with equal facility. If we look at NODEs and SLOTs as representing *position* only, i.e. a total ordering of events in the controller, then all four data-types act uniformly as *labels* that describe the grammatical characteristics of those positions. The motive behind

15. The constituent structure label *subargument* is properly a SLOT-FEATURE, not a CATEGORY-FEATURE as it is here, because it describes the relation of its contents to surrounding constituents at that level in the discourse rather than describing the structure of the contexts internally.
16. In the implementation of MUMBLE, CATEGORYs are distinguished from CATEGORY-FEATUREs only by being the first item on a NODE's features list.

singling out one of the features in the list as "more significant" is a heuristic judgement about the clustering of grammatical facts—that knowing just that one feature will tell you most of what you need to know about the NODE or SLOT, and that for purposes of casual inspections the other features can safely be neglected most of the time.

The meaning of a constituent-structure-label (the collective type for CATEGORYs, CATEGORY-FEATUREs, SLOT-NAMEs, and SLOT-FEATUREs) can be encoded either via direct references to the label's name or via expressions in the label's attached GRAMMAR-ROUTINEs—the subject of the next section. Direct references will always take the form of comparisons—identity checks—between a label's name (or a disjunctive list of names) and the value of a GRAMMAR-VARIABLE (or a LOCAL-VARIABLE derived from a GRAMMAR-VARIABLE). In MUMBLE, there are four GRAMMAR-VARIABLEs whose values are constituent-structure-labels: *current-node*, *current-slot*, *mother-node*, and *vertical-context*. The first three are self-explanatory, the last is a list of alternating CATEGORYs and SLOT-NAMEs which describes the direct path through the tree to the present position of the controller. It is used in calculations of parallel context. The variables are all maintained by the controller itself (non_specific_grammar_routines_n.y.w.).

4.2 Grammar routines

Form In MUMBLE, GRAMMAR-ROUTINEs are attached to particular constituent features by the same load-time operators as are used to specify the features themselves and their intrinsic-features, examples of which have already appeared in the last several pages. The routines themselves are written as LISP procedures: Each routine is given as a sequence of expressions enclosed in a lambda expression. The lambda assigns a local variable name to the input parameter supplied to the GRAMMAR-ROUTINE by the controller (page <arguments_to_grammar_routines>). Any of the operators that manipulate linguistic component data types may be used in the expression, as well as any PREDICATEs that test accessible properties of the tree or the discourse history. The LISP conditional (cond) is used to encode contingent or alternative actions, and the LISP "let" expression is used to define and provide initial values to any local variables that might be needed in addition to the input.

```
(define-schema if-then-schema (clause complex)
 slots (if-slot then-slot))

(define-label if-slot
 intrinsic-features (clausal coordinated-slot)
 enter-slot
    (lambda (contents)
         (mprint '|if|)   ;vertical bars delimit strings
         ))

(define-label then-slot
 intrinsic-features (clausal coordinated-slot)
 enter-slot
    (lamba (contents)
         (mprint '|then|)) )
```

GRAMMAR-ROUTINES employ a number of special operators. Mprint, used above, is the operator used to have words printed out directly without being first positioned in a SLOT or going through the morphology routine. Its argument is a STRING. Other special operators provide the capability to set and reset GRAMMAR-VARIABLES, to make GRAMMATICAL-DECISIONS, and to alter the contents of current-slot in various ways. These will be defined in succeding sections as those aspects are discussed.

Function When an ENTRY makes a choice of extended surface structure, it uses a PHRASE to create a path for the controller—a temporal pattern of events. If there were no postponed decisions embedded in this path, there would be no need for GRAMMAR-ROUTINES. This is because if there were no postponed decisions, there would be no later additions of new events to the path—all of the grammatically and rhetorically relevant events would already be present—consequently, any predictable morphological, lexical, or positional adjustments could be performed at the time the path is created. But of course, postponed decisions are central to this design. Consequently GRAMMAR-ROUTINES are essential as a means of actualizing the dormant effects of decisions that have already been made but could not be implemented because the projected events that they referred to were not yet present in the path of the controller.

GRAMMAR-ROUTINES act as soon as the target events appear such as when a comma is printed during a [coordinated-slot leave-slot] event. Their actions fall into four classes:

(1) direct additions to the text (via. mprint);

(2) maintance of the deitic aspects of the state of the linguistic component by setting and resetting the values of controller variables;

(3) embellishments to the tree in front of the current position of the controller, this (exaustively) includes:

  (1) making attachments to or ammending the DECISIONS of ELMT-INSTANCES,

(2) moving attached items off ELMT-INSTANCEs and into the tree as constituents,

(3) creating new NODEs and SLOTs as permitted by the rules of "latent constituent slots" discussed below,

(4) moving constituents already in place to "transformationally equivalent" positions in the tree;

(4) making grammatical decisions (which may, in turn, perform any of the above actions).

4.3 Grammar-decisions

A GRAMMATICAL-DECISION is a DECISION made from within a GRAMMAR-ROUTINE. Such DECISIONs exist (1) because many available options in the grammar have no direct counterpart in the concepts of the speaker or expert program yet still must be selected among, if only on indirect, heuristic grounds, and (2) because this is not always true—what is unmotivated in one domain may be motiviated in another—therefore, using DECISIONs to represent the criteria (rather than expressing them directly in the GRAMMAR-ROUTINEs as conditionals) provides the designer with a clean interface. Designers may change or augment the GRAMMATICAL-DECISIONs on the same basis as they write ENTRYs for a dictionary.

GRAMMATICAL-DECISIONs are used in conjunction with a special operator, *make-and-execute*, This operator acts much like the entry-interpreter: it makes the DECISION, evaluates its CHOICE, and makes a RECORD of the event for the discourse-history. The example below is typical in separating the preconditions of the decision from the designer-governed, enabling conditions.

```
(define-label  coordinated-slot

 after-realization
    (lambda (contents)
        (cond ((and previous-conjunct
                    (reduceable-category (category-of contents))
                    (same-msg-elmt (first-constituent-of previous-conjunct)
                                   (first-constituent-of contents)) )
               (decide 'reduce-category)) )))
```

The structure and manner of definition of a GRAMMATICAL-DECISION is the same as for a regular DECISION with one exception. As a matter of convention and designer discipline, GRAMMATICAL-DECISIONs and their CHOICEs have no parameters. This means that the only objects that they can effect are those pointed to by grammar-variables. Thus the CHOICE used below, *suppress-first-constituent*, implicitly means "suppress (i.e. replace with a TRACE) the first constituent of the NODE created as the result of the just finished realization".

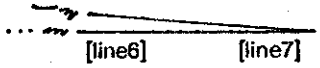
```
(define-decision  reduce-category

    default  (supress-first-constituent)
             ;i.e. do it unless there's a reason not to.
    ((member 'do-not-supress-first-constituent)
              (strategies-used-during  previous-conjunct)
     (do-not-supress-first-constituent))
    ((was-excessively-long  previous-conjunct)
     (do-not-supress-first-constituent))
    ((reduction-is-expressly-blocked)
     (do-not-supress-first-constituent))
    )
```

Like regular DECISIONS, GRAMMATICAL-DECISIONS can be modified by IMPURE-DECISIONS. These impure-decisions will typically be "pre-defined", i.e. created at system-load time rather than while the linguistic component is running. One such object, used by the DEFAULT-DECISION discussed just below, is *inhibit-contraction.*

```
(define-impure-decision  inhibit-contraction
 preempting-choice  (don't-contract))
```

4.4 Attachments and default-decisions

When an ENTRY makes a decision that cannot be implemented immediately because of references to linguistic events that are planned but do not yet exist, it still must make some addition to the tree so that its decision will be remembered and acted on later. This addition is typically an attachment to some ELMT-INSTANCE. Consider this example. In the proof on page <barber_proof>, we have a situation where an ENTRY that acts very early makes a decision that cannot be manifested in the text until the realization process has gone through *five* sucessive recursions. The initial situation is as below.



... Rule: Reductio-ad-absurdum
wff: negation106

We are at the last line of the proof, interpreting the ENTRY for the inference rule of *reductio-ad-absurdum.* Being, as it is, the conclusion of the proof, this ENTRY wants to somehow emphasize the difference in polarity between its formula ("A") and the first line of the proof, i.e. the assumption that has now been proven to lead to a contradiction ("not A"). However, all of the

linguistic CHOICEs that would "emphasize-polarity" depend for their selection on how the formula will be expressed in English—a datum which is obviously not yet available.

Faced with this predicament, the ENTRY does the best that it can do: it returns an instance of the formula as its immediate realization and attaches to that instance the MARK *emphasize-polarity* with the VALUE *negative* indicating the direction to be emphasized.

[line7]
negation106
emphasize-polarity negative

The ENTRY expects that some later routine (it does not know which one) will react to the attachment and carry out its intent.

MUMBLE's grammar has a strategy for emphasizing polarity, namely to use the "emphatic *do*"—"*Guiseppi does shave all those people*" for positive polarity, or for negative polarity to "stress" the word *not* it by printing it in capital letters and not contracting it. Since this is a general strategy, it is represented in such a way that it can be applied to *any* clause, without having to add anything to the ENTRY which created the clause, i.e. it is a DEFAULT-DECISION.

```
(define-default-decision  emphasize-polarity

  relevant-when  (has-feature  matrix 'clause)
  gating-condition
    (has-attachment  the-elmt-instance 'emphasize-polarity)

  ((eq (attachment-value  the-elmt-instance 'emphasize-polarity)
       'negative)
   (stress-the-negation  matrix))
  ((eq (attachment-value  the-elmt-instance 'emphasize-polarity)
       'positive)
   (emphatic-do_or_stress-copula  matrix)) )
```

This particular DEFAULT-DECISION is only relevant when the realization is a clause; yet in this example and many others, the ELMT-INSTANCE on which the attachment is made goes through several intervening realizations before the clause is realized. if the decision is going to be remembered long enough to be implemented, its attachments must be "passed along" from ELMT-INSTANCE to further embedded ELMT-INSTANCE, for which purpose every attachable TAG has a trivial DEFAULT-DECISION like this one:

```
(define-default-decision emphasize-polarity

  relevant-when (eq 'elmt-instance (type-of matrix))
  gating-condition (has-attachment the-elmt-instance 'emphasize-polarity)
  default (pass-through-attachment 'emphasize-polarity) )

(define-choice pass-through-attachment (mark)
  mode refiner
  extras (attach-to matrix mark (attachment-value mark matrix)) )
```

Of course, the whole point of "default" decisions is that they can be overridden when an ENTRY has a specific alternative. Such is the case in this example. The ENTRY for quantified formulas (pg.207) in the logic domain has its own DECISION with decision-name *emphasize-polarity*. Its choice of realization is to express the negation as the determiner "*no*" on the np that expresses the quantified variable: "*there is no barber...*". The presence of this regular DECISION preempts the DEFAULT-DECISION.

MUMBLE's grammar includes DEFAULT-DECISIONs for negation, polarity, adjuncts, tense and aspect, plurals, modifiers and qualifiers to np's, and intensive reflexives. What these phenomena all have in common is that, at the message level, they can all appear as *relational operators*. Their sources are not integral data-structures in the expert program but attitudes toward those structures adopted by the speaker. Consequently, their realization will be controlled largely not by their own ENTRYs but by actions parasitic on the ENTRYs of expert program structures. DEFAULT-DECISIONs, because they are transparent to regular DECISIONs, are thus a natural design choice for them.

4.5 HOOKs

Early in the course of this research, it became clear that the form of certain grammatical categories, in particular the verb group, was so intricate and so dependent on the contingencies of what items were or were not present that they were best constructed in two phases. First, as the items were decided upon they would be accumulated in a central location, organized relationally. Then, once all the items were present, the rules of the grammar would be applied and the appropriate constituent structure for that particular combination of items constructed. Subsequently, with the development of the 2-color hypothesis, it was found that in many cases the use of explicit constituent structure could be be avoided in favor of direct insertions into the text by the morphology routine.

The relational organization of extra-constituent items in the tree is provided by HOOKs. HOOKs are part of a PROPERTY-LIST mechanism comparable to the one used for the attachments on ELMT-INSTANCEs. Every NODE may have hooks property, consisting of any number of

HOOK—VALUE pairs. HOOK-VALUEs are always WORD-INSTANCEs or BOOLEAN. For example, the verb group involves the following hooks: *modal, have+en, be+ing, be+en, past*[17] *pre-aux1-adverb, post-aux1-adverb, pre-mvb-adverb,* and *post-mvb-adverb.* These may be placed on whatever NODE is available at the time the decision to use them is made, so long as that NODE is somewhere on the path from *the-root* to the verb group where it is to have its effect.

4.6 Transformations

When an ENTRY makes a matrix CHOICE, it itself has selected a certain choice of WORDs and subelements and a mapping of these onto roles in specific syntactic relations. If the CHOICE has a

property however, then those relations describe a *set* of possible constituent structures. Further decisions must be made, this time on the basis of grammatical rules and usage heuristics that are common to all of the ENTRYs in the dictionary. This commonality makes TRANSFORMATIONs an integral of the implementation of the interlingua between speaker and lingustic component.

TRANSFORMATIONs are one of the devices available to the designer for abstracting common decision-criteria away from individual ENTRYs and into a uniform, independent mechanism, thereby relieving the individual ENTRYs of the need to explicitly include them. They are based on the linguistic notion of equivalence classes of syntactic constructions [harris], or, as I will refer to them here, *transformational families.* The idea is that we can identify patterns of CATEGORYs and SLOT-NAMEs such that texts which employ them are, on that basis alone, uniformly subject to certain grammatically or intentionally induced variations.

For example, in MUMBLE, every CHOICE that specifies a PHRASE with both a [subject] and a [object1] ("direct object") is subject to TRANSFORMATIONs in this family:

unmarked: "*Macbeth murdered Duncan.*"
passive: "*Duncan was murdered by Macbeth.*"
passive_no-agent: "*Duncan was murdered.*"
nominal1: "*Macbeth's murder of Duncan.*"
nominal2: "*Duncan's murder by Macbeth.*"
nominal2_no-agent: "*Duncan's murder.*"

Other transformational families that have been developed are embedded clauses with [subject] and [predicate], extraposition from [subject] ("*Pleasing John is easy.*", "*It is easy to please John.*"), the genitive—possive alternation in noun phrases and alternate orderings of constituents in the verb phrase, both under the control of a new information—old information distinction. (Questions, relative clauses, clefting, and right and left dislocations, and ripping or copying rules in general are not analyzed as transformations. See section <control_vs_unbounded_movement>.)

17. These last four HOOKs have BOOLEAN VALUESs. They are part of a very low-level analysis of the verb group. Higher-level descriptions such as *narrative-past* would be translated into these terms by dictionary subroutines (see interlingua_verb_group_n.y.w.).

Selecting TRANSFORMATIONS The value of a CHOICE's transformations property is a relatively complex object created from the TRANSFORMATIONAL-FAMILYs that apply to the CHOICE. The TRANSFORMATIONAL-FAMILYs have one important property:

conditions A list of one or more CONDITION-SETs

Below are the conditions of the [subject]—[object1] family as they were used for the example in the introduction.[18] The PREDICATEs these CONDITION-SETs employ fall into three groups: (1) those that test properties internal to the CHOICE-APPLICATION ("will the [subject] be filled?"), (2) those that test *thematic relations* that apply to the message elements involved ("are we talking about Duncan or about Macbeth?") and (3) those that test the larger context in which the instantiated PHRASE will appear ("will it be an independent sentence, a [subject], a [complement] ?").

```
((slot-will-be-empty 'subject)
 (not (is-true 'possible-sentence-start))
 (viewed-as-an-event the-elmt-instance)
 (nominal2_no-agent))

((slot-will-be-empty 'subject)
 (is-true 'possible-sentence-start)
 (passive_no-agent))

((eq (unmarked-slot-for-argument (argument-in-focus))
     'object1)
 (passive))

((eq (unmarked-slot-for-argument (argument-in-focus))
     'subject)
 (unmarked))
```

A brief inspection of this family will reveal (1) that it has a very real ordering (the last two thematic transformations make no sense if the constituent is only going to have one noun phrase), and (2) that it has obvious gaps (what happens, for example, in the case where passive is applied in a non-sentential context). These two aspects are sorted out when the CONDITION-SETs of a CHOICE's applicable TRANSFORMATIONAL-FAMILYs are merged to create its transformations property. The rules for inter-family ordering, for example that a "thematically" controlled TRANSFORMATION like passive must be allowed to apply before a "grammatically" controlled one like equivalent-noun-phrase-deletion, are encoded *ad-hoc* directly into the postprocessor that

18. You will notice that there is no CONDITION-SET that selects either "nominal-1" or the full version of "nominal-2" above. Even though they are a part of the family linguistically, the conceptualizations available in the Macbeth domain were inadequate pragmatically to pick out those TRANSFORMATIONs for any reason other than variation for variation's sake. This is a common problem for the designer. The best solution is probably always to omit lingustic alternatives that the domain cannot justify rather than to include them and risk the program saying something that it did not mean.

builds the transformations properties.

The object in a transformations property is similar to a *transition network*. Its starting state is the CHOICE it is a property of. Transitions between states are defined by discrimination nets constituted from the PREDICATEs of the CONDITION-SETs of the TRANSFORMATIONAL-FAMILYs that apply to the CHOICE. Each state is assigned one TRANSFORMATION. When the state is reached, its TRANSFORMATION is applied. Any necessary adjustments to the set of applicable TRANSFORMATIONAL-FAMILYs brought on by the application of a particular TRANSFORMATION are already taken into account in the descrimination net leaving that node.

This encoding technique obviates the need to search the grammar in order to apply transformation, the "search" having been done at system-load time when the transformations properties are compiled.

Applying TRANSFORMATIONS

A TRANSFORMATION is a specification of an "editing" procedure that is applied to the properties of a CHOICE. In MUMBLE, edits to the phrase property are given as a list of operations (all implicitly parameterized by the original xpr(phrase)) and edits to the map as a mapping from the SLOT-PATHS of the old map to new SLOT-PATHS. For example, below is the transformation that subordinates clause complements and adjuncts whose subject's are the same as the subject of the main clause.

```
(define-transformation  equi-np-deletion

 phrase  ((return-only-the-predicate))
 map  (((subject) . nil))  )
```

It is quite simple: all parts of the old phrase except for the specified contents of the [predicate] are to be excised, and the item in the map that specified what parameter was to be positioned in the [subject] is to be removed. The definition of passive with agent is more complex, calling for adjustments to the specified CONSTITUENT-SCHEMAS:

```
(define-transformation  passive_with_by-phrase

 phrase  ((adjoin_by-phrase_to_vp)
          (original_vp_submerged_into_an_adj-comp))
 (map (((subject) . (predicate pred-adj by-obj))
       ((predicate mvb) . (predicate pred-adj mvb))
       ((predicate object1) . (predicate pred-adj object1)))  )
```

The accompanying diagram illustrates what this TRANSFORMATION does using constituent trees.

Remember, however, that the transformation-interpreter actually manipulates symbolic expressions and that the constituent structure is only created once all of the TRANSFORMATIONS have been applied.

TRANSFORMATIONS may make three kinds of adjustments to the patterns of constituent structure that the ENTRYS select:

(1) prune away all constituents but one;

(2) embed the whole phrase or one of its constituents within a new structure substituted for it in the same location (as done above with the VP);

(3) "vitalize" a latent constituent slot (see below).

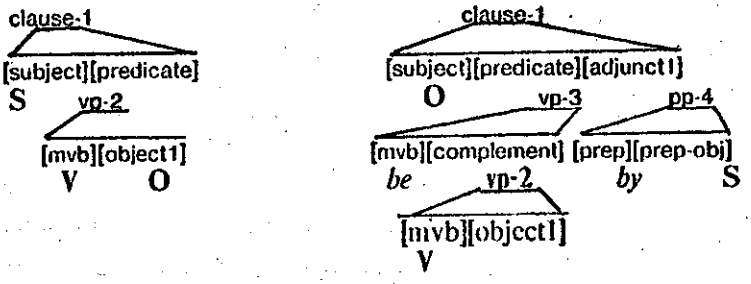
These same adjustments can also be done by GRAMMAR-ROUTINES. See, for example, the discussion of heavy-phrase shift (heavy_phrase_shift_n.y.w.), "predicate lowering" (III.3.7), or the filter on verb particle placement (particle_filter_n.y.w.).

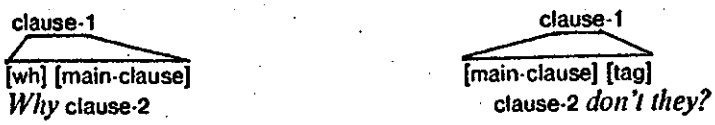
4.7 Fine points

Dynamically created patterns of constituents We have seen how in this theory the design of the constituent structure has been made to reflect the "chunk-size" of elements in the message. Another important, if more subtle, consideration is timing. Consider: every clause has a main verb and, depending on what the verb is, some number of objects. One of the reasons for not having the CONSTITUENT-SCHEMAS for clauses always include the the verb and its objects is that oftentimes when it is known that a clause is needed, it is not known what its verb will be and therefore what pattern of objects will be needed. By dividing the clause into a [subject] and a [predicate] we can postpone the decision about the objects until the verb becomes known.

There is also, of course, the question of modularity and generality in the representation. Verb phrases can appear outside of clauses; consequently independent CONSTITUENT-SCHEMAS for them will always be needed. If the grammar included a "flat" schema for every clause as well as one composed from some verb phrase schema and a "basic" clause schema, its size would grow considerably but with little increase in real information. This is not an issolated phenomena. A "basic" clause can be part of a WH or a tag question,

---





do hobbits have hairy feet? *Hobbits have hairy feet,*

noun phrases can have trailing apposatives or intensive reflexives, basic clauses may have any number of leading or trailing adverbial adjuncts—most any adjective can have a complement ("*murdered by Macbeth*")—and so on.

When the generalization involves a nesting of categories, it can be dealt with by using PHRASEs, as with basic clauses and the different kinds of verb phrase. The question remains of what to do when (1) the newly needed constituent is at the same level as an established schema, and (2) the fact that it will be not needed is not known when the original schema is created but only becomes known later. Here there are two choices open to the designer: either all CONSTITUENT-SCHEMA are maximal to begin with (this will lead to a multiplication of the number of schema in the grammar even though most of the time many of their constituent slots will be empty), or, SLOT-NAMEs may be added to already instantiated schema (this is perhaps unaesthetic).

I have opted for the second design. There are "constituent adding" operations in MUMBLE's grammar that "activate" conceptually *latent slots*. *Add-by-phrase-to-vp*, used by the passive TRANSFORMATION, is one such operation. At this writing, there has yet to be a declarative representation designed to describe what slots are "latent". Instead, the procedure that adds the new SLOTs to constituent structure directly incorporates those rules that have been identified.

Where do verbs go? Open-class, "content" words originate in ENTRYs. If this is all that matters, then it is for the most part a matter of taste whether verbs appear there as arguments to CHOICEs or as part of the CHOICEs themselves. That is, one could either write:

"(clause-one-object '*murder* 'macbeth 'duncan)"

or

"(murder-clause 'macbeth 'duncan)"

where *murder-clause* would be defined as:

```
(define-choice murder_clause (s o)

phrase (basic-clause ()
          predicate (vp_object1 ()
                       mvb murder))
map ((s . (subject))
     (o . (predicate object1))) )
```

Certain things do follow differently from the two alternatives. The "verbs as arguments" design requires fewer CHOICEs altogether. However, this should not be of any concern I think. As a practical matter, most of the CHOICEs that a "verbs as choices" design calls for will be almost identical and can be written quickly through the use of macros; in addition, the overall design is already heavily biased toward "space expensive" design alternatives so as to gain speed at run-time. More to the point perhaps, there is already an independent need to keep a history of what CHOICEs have been made. The "verbs as arguments" alternative would require increasing the size of this record enormously to accomodate recording the CHOICEs with each of their arguments, while the "verbs as choices" alternative would require no change at all. In addition, there is much to be said for holding to the discipline of designing operations so that their logical properties are not changed when their arguments change. Automatic reasoning (such as the symbolic analysis of an ENTRY's alternatives of the design of the postprocessor which constructs transformations properties) is greatly simplified when the arguments in an expression may be safely ignored.

Encoding facts into the SLOT-NAMEs versus finding them in the context Earlier examples have referred to single [complement] slot, rather than to an [infintive-complement] plus a [report-complement] plus a [for-to_complement], etc.. In so doing, they have reflected a design decision about the relation between a SLOT-NAME, its associated GRAMMAR-ROUTINEs, and the context they appear in.

In grammars of English, the term "complement" covers aspects of both form and function. On the one hand, a complement is some argument to a verb (or to an adjective) that is not a noun phrase. Thus we have "subject complements": "*Macbeth grew unhappy*", and "object complements": "*They considered him unworthy*", depending on whether the complement tells us something about the subject or the object. On the other hand, complements are also named according to their form. We have "nominal complements": "*Macbeth became king*", "prepositional complements": *John kept in great shape*", and so on according to the category of the complement constituent. What does this nomenclature mean in production terms and how should SLOT-NAMEs reflect it?

One of the functions of a SLOT-NAME is to directly influence the decision-making of the ENTRYs of the message elements it will contain. Thus the point of using a *prepositional-*

complement would be to use CHOICE-FILTERs to "force" the message element to be realized as a prepositional phrase. The experience with MUMBLE, however, has been that such forcing has not been necessary or was even detrimental, i.e. whether the complement is an noun phrase or an adjective phrase is often irrelevant to the ENTRY that chooses the CONSTITUENT-SCHEMA that positions it—the ENTRY may only care that it is a description of the [subject].

A further role of a SLOT-NAME is to describe how the element its slot contains is related linguistically to the other elements in the immediate context. For example the TRANSFORMATION that deletes the (potential) subjects of subordinate clauses if they are identical a noun phrase in the main clause will function differently depending on whether a subject or an object complement is involved. Even here though, the information need not be encoded in the SLOT-NAME. MUMBLE's grammar uses a single SLOT-NAME, *complement*, choosing to encode the specification of subject versus object as a property of the main verb. Instead of specializing the PREDICATEs of the relevant TRANSFORMATIONAL-FAMILYs to SLOT-NAMEs, they are made to test for a common property on the main verb (readily available as the controller variable *current-verb*). Its value is the GRAMMATICAL-DECISION that will select among the possible TRANSFORMATIONs for that type of complement.

4.8 Controller variables

The extended surface structure plays two roles: (1) it is a record for the controller of the actions that the ENTRYs and GRAMMAR-ROUTINEs have planned; and (2) it defines the *linguistic context* in which any further decisions will be made. The role of the deitic variables maintained by the controller is to structure the linguistic context so as to simplify the problem of reasoning about it. By using variables to point directly to facts about the extended surface structure as soon as they appear, the active procedures of the grammar are relieved of any need to search the structure by scanning or pattern matching.

Legitimate values

The values of controller variables may range over: constituent-structure-labels ELMT-INSTANCEs and WORD-INSTANCEs, RECORDs in the discourse history, and booleans (used to mark progress through the tree, see below). NODEs or SLOTs are not legitimate values. The value a variable takes is dependent (obviously enough) on the point where it is set. Consider the case of the variables in the record of a clause.

```
(define-label basic-clause
 enter-node
   (lambda (the-clause)
        (shallow-bind 'current-subject ;the name of the variable
                      (getslot 'subject the-clause) ;calculation of its value
                      the-clause) ;the node whose record this is added to
   ))
```

When [basic-clause enter-node] runs, the contents of the [subject] will be an ELMT-INSTANCE. This is exactly the right level of representation at which to refer to "the subject" in the grammar. Practically all rules which refer to the *current-subject* (e.g. reflexives: "*anyone who shaves himself*", and embeded clauses: "*Macbeth wanted to be king*") use it in an identity test, e.g. comparing the *current-subject* with a subelement of the message element that leads to the embedded clause. Since this test happens at the message level while the transformations are being applied, it is simply a matter of checking if two ELMT-INSTANCEs are instances of the same msg-elmt. One can argue that for doing number-agreement, the optimal value for *current-subject* is the final contents of [subject], rather than the initial. However, if the interface function elmt-pluralp is implemented as a "marked—unmarked" relation, i.e. plural if known to be plural, otherwise singular, then there would be no extra benefit to having the final value.

```
(define-label mvb    ;"main verb"
 leave-slot
   (lambda (the-verb-group)   ;see III.2.4
        (shallow-bind 'current-mvb
                      (4.2c)(getslot 'main-verb the-verb-group)
                      current-clause)
   ))
```

On the other hand, it is the lexical form of the main verb that is grammatically important rather than the message element that lead to it. Consequently, the proper time to fix the value of *current-mvb* is when we can be sure that the contents will be a word. (It may occur to the reader that, because *current-mvb* is not set until so late in the clause, it will not be available for use in questions for subject-verb inversion. This is indeed the case and a limited form of lookahead can be required. See section subject_aux_inversion_n.y.w..)

Grammatical relations over variables rather than tree-structure

The definition of the grammatical notion of, e.g., "subject" as variable maintained by the shallow-binding discipline of section shallow_binding_n.y.w., rather than as a relative position in a tree or a p-marker, has unexpected side-effects: i.e. *current-subject* retains its value *after* the controller leaves the region of the clause that contains it. It will change only when the next clause

is entered. It happens that this "side-effect" conforms nicely with the constraint that the chunk-size of the message is the determining factor in the design of constituent structure.

Consider the definition of a "sentence" in MUMBLE's grammar. The notion of a sentence is a strictly linguistic one with no counterpart in the data-structures of an expert program.[19] Consequently, sentence boundaries will be placed by the grammar rather than by ENTRYs. In MUMBLE, this is done in the following way. Part of [*always* enter-node] monitors the passage of the controller through the tree for the first occurance of a NODE with a syntactic CATEGORY (e.g. not *discourse*) on the path down from *root node*. The record of this this NODE is made the value of *current-sentence*. When this NODE is finally left, [*always* leave-node], noticing that fact, turns on the boolean grammar-variable (see below) *potential-sentence-start*. The morphology routine is sensitive to this variable and when it is *true* will preceed the next word it receives with sentence-terminating punctuation. That is, the visible manifestation of *potential-sentence-start* does not occur until the first word of the next sentence (unless that is the end of the message).

This delay provides a period of limbo during which the ENTRY for the right-sibling of the just finished sentence or a GRAMMAR-ROUTINE associated with its slot has the option to decide that the realization for that sibling elmt-instance should be adjoined to the previous sentence rather than starting a new one. (C.f. example in the introduction, page <merging_becomes_king_n.y.w.>. To do this, it has only to set *potential-sentence-start* to *false* and, of course, initiate the appropriate adjunction transformation. The end of the sentence will thus be "postponed" until the siblings realization is complete.

During the limbo period, clause-level grammar-variables retain the value they had, even though their clause has been completed (in terms of tree structure). This is because the shallow-binding algorithm, realizing that there were (at that time) no further constituents in that clause, did not have to rebind them to their former values—they had no "former" values. Because the values have not changed, the subordination transformations (pg.105) can freely refer to *current-subject*, just as though the adjoining message element had been another constituent of the clause.

Flags for grammatical events

Certain of the rules of English that are the responsibility of the morphology routine, including the expression of tense, the position of the first auxiliary verb in questions, and the definition of sentences under the above analysis, are associated not with particular words but with certain events defined by the extended surface structure. In order to inform the morphology routine of when these events occur, MUMBLE uses grammar-variables with boolean values—"flags". The GRAMMAR-ROUTINE associated with the event, for example [mvb enter-slot] (pg.<mvb_enter_slot_n.y.w.>) or [subject enter-slot] (pg.<subject_enter_slot_n.y.w.>), set the

19. A speaker, on the other hand, will doubtless have heuristics about the proper length of a sentence and of possible trade-offs in the decomposition of a discourse into sentences. See [gretchen_thesis] for some discussion.

appropriate variable to *true*. The morphology routine, which is constantly monitoring for these variables, notices when they become *true*, reacts according to its rules, and then sets them to *false*.

5. The Interface

This section defines the relationship of the linguistic component to the other components of the system. It discusses the merits of couching messages in the same representation as used in the speaker and expert-program, and delineates which aspects of a message's structure are critical for its use by the linguistic component. Interface functions and ELMT-INSTANCEs are introduced as technical devices which allow the component to manipulate externally defined objects without having to know how they are implemented.

i. Invoking the linguistic component

To the rest of the system, the linguistic component is a subroutine—a subprocess which the speaker activates when necessary to perform a specific task (and which cannot function except when explicitly activated). Once activated, the linguistic component runs independently of the rest of the system in that it is not designed to be either monitored or interrupted by outside processes. However, parts of the linguistic component (specifically the PREDICATEs and ACCESS-FUNCTIONs used in DECISIONs (pg. <computational_environment>)) need access to the computational state of the speaker and expert program.

The form of the task is always the same: "take this specification—the message—and produce the corresponding English text". The message is passed to the linguistic component as its one explicit parameter, and the text produced as a side-effect. No importance is placed on the value which the linguistic component returns to the process that activated it, as any actions or computations of long term significance will have been already entered by side-effect into the discourse history (pg.<discourse_history>), a data base maintained jointly with the system's language comprehension facilities.

There is no presumption that calls to the linguistic component should result in texts of any one fixed size. Single calls to MUMBLE have, for various micro-speakers, produced texts ranging from single exclamations to multi-page texts. Nor do single calls have to be equated with complete turns in the conversation, as the linguistic state of the component is preserved between calls and the text can be "picked up where it left off". (See II.2.3.)

5.1 Using the speaker's own representation

The linguistic component is a *transducer.*[20] It takes an expression given in one medium—the internal representation of the speaker, and creates a corresponding expression in another medium—English text. To be able to do this, the linguistic component must (1) understand the structure of the language in which the input specifications (messages) are couched, and (2) have some way to determine the correspondences between the individual elements of the messages and English phrases.

These requirements imply that the linguistic component has a great deal of speaker-specific knowledge, yet, at the same time, the design criteria require that the component be a transportable module that can be used by many different speakers without major modification. This means that it can have no intrinsic preconceptions about the form of a speaker's representation—it must have some way of holding message elements "at arm's length" while it decides upon their English realization.

One way to accomplish this might be to require that all speakers use the same message language, i.e. a common set of representational conventions, common non-linguistic concepts, and, by implication, a common ontology. A single dictionary would be written expressing the correspondence between that common message language and English. This approach is the one that has been adopted (at least implicitly) by virtually all of the other language production programs developed thus far, because they are all designed to work from only one kind of underlying representation (see [ddm_past]).

It is an inescapable fact that the range of representations presently in use in expert programs is incredibly large (see the survey in [Ron_&_Brian]). Thus if one linguistic component is to serve multiple expert programs, some accomodation must be made; the question is where. One must ask whether the effort of translating from the representation of a new expert program to a common message language would be any smaller than that of translating directly to English, avoiding the middle-man.

I have elected to avoid the middleman. When a speaker assembles a message, it will be in terms of the actual data structures used in the expert program. To make make these structures inteligible to the linguistic component, a dictionary must be written for each new domain that the linguistic component is to be combined with. The dictionary gives the correspondence between the new message representation and the linguistic component's uniform language for describing (constructing) English phrases. Putting this another way, we can look at a speaker's messages as sentences in an ideosyncratic *message language* where the dictionary defines the language's semantics, determining its interpretation into texts.

---

20. "Transducer" is used instead of "translator" because of its more appropriate connotations. Transduction can involve very dissimilar media, whereas translation is between tokens that are both expressions in some language—implying a commonality of structure that is not assumed here.

The dictionaries for two different domains will be similar to each other (i.e. will share components) to the same extent that their two representational systems and ontologies are similar (i.e. have the same form and are intended to be rendered into English in similar ways). The structures employed by different expert programs are, of course, designed for problem solving in different real world domains: medicine, symbolic mathematics, programming, etc., and consequently will necessarily correspond to different open-class vocabularies. However, the ways in which two systems choose to talk about what they know—structures special to their *speakers*—may well have much in common. In particular, certain relations such as: "modifies", "follows", and "conjoined with" (which are a part of the linguistic component's internal vocabulary) appear to also be a natural part of the message-level vocabulary of the speakers that have been studied. Such relation may well constitute an *interlingua* ultimately common to both parts of the language production process. (See also section <beginnings_of_an_interlingua>.)

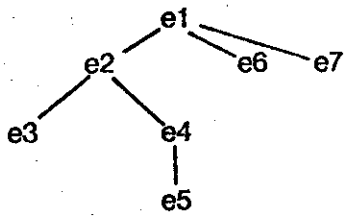
The resulting picture of messages is that they are comprised of a combination of data structures from the expert program (some possibly assembled just for the purpose of that message) and symbols known to both speaker and linguistic component. The syntax can be whatever is convenient for the speaker.

5.2 Messages

Messages are expected to be compositional structures over a common vocabulary that the dictionary can interpret. The composition can be expressed in an arbitrary syntax because the linguistic component does its decompositino of messages through *interface functions* implemented especially for each speaker. A convenient, unassuming abstract model for message structure is a tree of composed n-airy relations, as shown schematically in the figure below.

Each individual relation into which a message can be decomposed will be called a *message element.* In the figure, e1(e2, e6, e7), e2(e3, e4), e3, xe4(e5), etc. are each elements of the message; the *subelements* of xe1 are e2, e6, and e7, the subelements of e2 are e3 and e4, etc.. Message elements (and messages themselves) will be treated as members of a common data type, MSG-ELMT. Because they are objects defined and maintained by an entirely separate program, the properties of their data-type are of necessity specified indirectly, i.e.

An object is a MSG-ELMT if and only if it can be given as valid argument to all of the



interface functions.

This is the *only* formal restriction on their structure or mode of implementation though there are, however, "informal" restrictions which will determine how easy it is to develop a dictionary (see section <designing_an_interface>).

i. Message-element enumeration order

The linguistic component has been designed as a *serial* rather than a parallel process. As a consequence, only one message element will be realized at a time. The order in which the elements of a message are realized will be termed its *enumeration order.* This order is total and fully determined at the time the message is passed to the linguistic component (assuming, of course, that the speaker and expert programs do not change state while the component is active).

An enumeration order is not a property of the message *per se* but of the dictionary ENTRYs that will process it. This is because it is the individual ENTRYs that determine which potential subelements will be realized and which will not; furthermore, these decisions will be contingent on earlier decisions and on the linguistic context. For example in the figure, if the decision for e1 is to realize the subelements of e6 and e7 as sequences, then the enumeration order would be the same as the indexes. If, on the other hand, e6 and e7 were instructions describing how e2 was to be realized (as in the example in the introduction), then the order might be (e1 e6 e7 e2 ...).

The enumeration order of correctly assembled messages will meet a *well-formedness condition.* This condition will be stated formally in section well_formedness_condition_n.y.w. after more of the theory has been presented. Informally, it requires that the realization decision for each message element must be made early enough, compared to those of the elements to which it is related, that it can be implemented without having to change any of the decisions that have already been made. If satisfied, this constraint guarantees that the linguistic component will be able to find a realizing text for the message that is grammatical and that expresses all of its elements.[21] It does not, and cannot, guarantee that the text will be beautiful prose—that is the responsibility of the dictionary.

An algorithm for testing messages against the condition can be written; however, because of the extensive search required and the fact that the algorithm cannot provide corrective advice when deviations are detected, speakers are not expected to use it except as a theoretical device. Instead, I expect that a straight-forward set of conventions can be devised to govern message construction that will have the same effect; these will be discussed throughout succeeding chapters.

---

21. within the limits imposed by the dictionaries ability to handle island constraints, see planning_by_the_speaker_obeying_island_constraints_n.y.w.

5.3 Interface functions

Domain-specific information that the linguistic component needs to know about a message or a message element is acquired through a specific set of *interface functions* written specially for the domain involved. The interface functions are functions from objects defined by the speaker and expert program to objects defined by the linguistics component. Their design and their operation with various domain representations is taken up in detail in section issues_at_the_interface_n.y.w., and sample definitions from the micro-speakers are given in appendix VII.B.1.3. At this point, I will only list them individualy with a brief description of their use. All but one are functions of one argument of type MSG-ELMT.

entry-for Returns the dictionary entry that is to be used for that msg-elmt.

entry-arguments-for Returns the msg-elmt(s) that are to be assigned to the entry's parameters when it is processed. Entrys are typically shared by entire classes of msg-elmts and consequently may require access to the individual instances for particularizing information.

msg-elmtp A type-predicate. It tests whether or not its argument is of type MSG-ELMT.

same-msg-elmt Takes two arguments, each a MSG-ELMT or an ELMT-INSTANCE, and tests whether they are (or represent) the same object.

elmt-pluralp Tests whether the element will need to be marked plural if it is realized as a noun phrase.

elmt-gender Returns one of the SYMBOLS: masculine, feminine, neutral-gender.

elmt-reference-type Used by the pronominalization routine. Returns one of reference, description, or other.

distinguishable-kind Also used by the pronominalization routine. Takes two message elements as its arguments and determines whether they are members of disjoint conceptual categories that will not be mistaken for each other if pronominalized.

elmt-discourse-history, set-elmt-discourse-history Access functions associating msg-elmt with descriptions of how they have been used in the text thus far.

In addition to these access functions, an interface includes a *domain-evaluator*, that must be able to evaluate an expression with respect to the computational environment of the speaker, even though it will be called from inside the linguistic component.

Data type: ELMT-INSTANCE

Much of the linguistic component's power comes from being able to manipulate individual message elements: to embed them within linguistic structures, to have them decomposed into subelements, to look at their properties, to mark them for individual linguistic roles, and to record their linguistic history. But, because the implementational structure of a message element will vary from domain to domain, what the component actually manipulates are "stand in's" for message elements—objects of type ELMT-INSTANCE.

The use of ELMT-INSTANCEs has several benefits: (1) it promotes the transportability of the linguistic component as a whole; (2) it provides a way to mark a message element for special effects and to record its linguistic history without simultaniously affecting the "real" data structure within the expert program or presuming upon the expert's choice of implementation; and (3) it makes it possible to distinguish the separate occasions when the same message element appears in a text, i.e. each is represented by a different ELMT-INSTANCE and thus may have distinct markings and history.

The properties of ELMT-INSTANCEs reflect their functions. The first three properties are set when the instance is created. Their information is redundant (i.e. it could be read from the message element) and is here as a clerical convenience (although it can be edited to achieve certain special effects, see, for example, pg.<variable_entry_n.y.w.>). The rest of the properties, however, represent optional new information which will override or embellish the old. They are the means by which instances may be marked to achieve ideosyncratic effects.

real-msg-elmt The MSG-ELMT to which the instance corresponds.

entry-for The ENTRY for this instance. The value of the entry-for property of the instance's real-msg-elmt.

entry-arguments-for Similarly, a copy of the property of the same name on the entry.

decisions-to-make Again, typically copied from the entry's own version, but it may be changed to achieve various effects: see meta_entrys_n.y.w..

new-decisions A list[22] of zero or more DECISIONS not already included with those of the entry, and which are interpreted after the entry's own decisions.

amended-decisions A list of zero or more IMPURE-DECISIONS. These correspond by name to DECISIONS of the entry and augment or override parts of their actions.

22. As used here, the critical property of a *list* as a data-structures is that it is a sequence whose items are in a fixed order. Nothing is presumed about how the "sequentialness" of the items is determiner.

attachments A PROPERTY-LIST[23] of zero or more mark—object pairs. Each pair will be referred to as an *attachment.* The set of possible marks is determined by the analyses employed in the dictionary. The ttachments property is a facility for "marking" individual ELMT-INSTANCEs in a way which later can be recognized and acted upon by some DECISION or GRAMMAR-ROUTINE.

target_elmt A CONDITION-SET tested when the instance has been created in order to insure its correct positioning within the tree.

CONDITION-SETs DECISIONs and IMPURE-DECISIONs are all defined later in the discussion of the realization procedure. GRAMMAR-ROUTINEs are introduced in section <attached_procedures>.

ii. Creating instances

ELMT-INSTANCEs are created for MSG-ELMTs only when they are needed. This will be either when the element is about to be added to the tree, or when it must be marked in some way before it has been added. The first case is the most common. MSG-ELMTs only enter the tree by being arguments to CHOICEs, consequently, the work is done within the choice-interpreter (pg. <choice_evaluator>) and the new ELMT-INSTANCE is immediately stored in its intended place in the tree.

The various procedures that mark instances accept as their arguments either ELMT-INSTANCEs or MSG-ELMTs. If given the later, they will immediately create an ELMT-INSTANCE provided no such "early-instance" corresponding to that MSG-ELMT already exists. ELMT-INSTANCEs created at such times have no natural storage place and are kept with their MSG-ELMTs on an ASSOCIATION-LIST[24] reserved for that purpose. Subsequent references to "the instance" of that MSG-ELMT are routed to this instance rather than causing another to be created. Such msg-elmts that have been referenced before they would otherwise have appeared in the normal enumeration order are called EARLY-INSTANCEs.

ELMT-INSTANCEs are removed from the list when they are finally added to the tree. This removal occurs if and only if the CONDITION-SET of the target of the ELMT-INSTANCE is true at that time, otherwise, a new one is created. These provisions for EARLY-INSTANCEs are necessitated by the incremental realization constraint (pg.<incremental_realization_n.y.w.>) and extensively discussed in section problems_with_incremental_refinement_n.y.w..

---

23. A PROPERTY-LIST is a means of associating an arbitrary number of objects referred to as "values" with one object by pairing them with symbols referred to as "tags" Given an object and a tag, the associated VALUE may be retrieved. However, one cannot retrieve an object given a tag—value pair. This is the same notion of "property list" as used LISP except that here there is no significance to the linear ordering of the items on the list.

24. This is the same use of "association list" as in LISP except, of course, that linear order within the list is not relevant. ASSOCIATION-LISTs behave like variables in that they are named—one adds items to a specific list—and they are bound to a specific part of the process. In this case, the list is bound to the ELMT-INSTANCE creating process itself and therefore cannot be contextually rebound.

6. Realization

Linguists in the Prague and Firthian schools speak about "*realizing*" a concept as a phrase in a natural language. They use the verb "realize" in preference to the possibly more common verb "generate" because of an important difference in connotation. The term *generate* is used by people in formal language theory and transformation generative linguistics and has acquired a connotation as a "blind" enumeration of sentences, guided only by the structure of the grammar. *Realize*, on the other hand, is intended for the special (some would say more natural) circumstances of producing a particular phrase for a particular concept in a particular pragmatic and intentional context.-

6.1 Possible Realizations

I will use the noun "realization" with a studied ambiguity. It will mean either (1) the result of the application of the realization procedure to a msg-elmt, or (2) the English text that corresponds to some message element in some context. In other words, the text which realizes a given msg-elmt (meaning 2) is the transitive closure of the *to realize a msg-elmt* relation (meaning 1).

The realization procedure can return any of three kinds of objects: elmt-instances, nodes, and word-instances. The first two have already been discussed. Here is how "words" are treated, along with the treatment of "traces" which are treated by the controller as thought they were words.

Data type: WORD

English words are represented by objects of type WORD. All such objects have the two properties below, and some of them, particularly the verbs, may have additional properties that further describe their morphological structure. MUMBLE is capable of typographic output only, and consequently has no representation of phonemes, syllabic structure, or intonation.

Pname A string of characters in mixed case. It is always the word's primary typographical manifestation—the "root" form. The term derives from "print name".

Features A list of zero or more objects of type WORD-FEATURE. The total set of such objects is determined by the grammar.

The exact form that a word takes when printed is determined by the morphology routine (<the_morphology_procedure_n.y.w.>) which uses the WORD's pname as the base form and applies a uniform set of spelling rules to specialize the pname according to the grammatical

context and the grammatical properties of the WORD as specified by its features. Unpredictable irregularities in conjugational or declensional paradigms are flagged by optional properties whose values are the appropriate character strings (e.g. the string "h a d" is stored as the past-tense of "*have*").

There is no restriction that the pname of a word necessarily consist of only one "word" in the usual sense. WORDs may be as complex as the designer chooses. For example,in the vocabulary of the Macbeth domain, "*is a character in*" could be one word (presumably an intransitive verb). However, whatever is represented as a WORD will be treated as one, i.e. the morphology routine will manipulate it as *one* unit. For example, if it appears in the position of a main verb, tense or agreement will be affixed to the end of its pname's string. This could be catastrophic, e.g. "*Rosencrantz and Guildenstern is a character in Hamlet*".

Notice that there is no provision for any "semantic" annotation of WORDs. One cannot look at the properties of a WORD and tell what message element classes it could be a realization of. Such backpointers do not exist anywhere in the linguistic component. This is because (1) they would be impossible to construct in a transportable component without a common message language; and (2) they are not needed, because individual WORDs are selected directly as part of preexisting CHOICEs <choices> and not as the result of any sort of pattern-matching process.

There is also no concept of an "ambiguous word" in the linguistic component. There may well be several distinct WORDs tht have the same pname, but there is not any way to determine for a given pname what those WORDs are.

Whenever a WORD is selected as a realization, a new WORD-INSTANCE corresponding to it is created and placed in the tree for it. All of a word-instance's properties are copied directly from its WORD. In this text-based version of the linguistic component, the role of such WORD-INSTANCEs is largely technical: it is convenient in the implementation of MUMBLE's discourse-history to treat WORDs in the same way as MSG-ELMTs. However, in a speech-based system, it seems likely that WORD-INSTANCEs would be needed as a place to ground per-instance markings governing relative stress, duration, and pitch contour.

Data type: TRACE

As their name suggests, TRACEs are remnants or records of other objects which formerly held (or would have held) their position in the constituent structure. The notion of a "trace" was introduced into linguistics by Chomsky [chomsky_traces_first][chomsky_traces_best] as part of "annotated surface structure". My use of the term is not significantly different than his.

TRACEs are operationally equivalent to WORDS, and have the following two properties:

Pname Identical in specification to the pname of WORDS. Most TRACEs have a pname of zero characters, i.e. they do not appear in the output text.

Link The object that the TRACE has replaced.

The generic objects to which TRACES correspond are certain CHOICES, i.e. certain realization decisions. See section II.6.2.iv.

Data type: PHRASE

A PHRASE is a *specification* of a linguistically annotated tree structure whose nonterminal nodes will be instances of grammatical categories and whose terminals will be either ELMT-INSTANCES, WORD-INSTANCES, or TRACES. The trees that phrases specify have all of the formal properties which are normally associated with constituent trees in formal language theory even though they are not defined by a generative grammar. In particular, the "is a constituent of"—"dominates" relationship is defined, and there is a unique, total ordering of the tree's fringe. [[??provide a formal exposition and definition (perhaps later) ??]]

When a PHRASE is selected as the realization of a message element, it is *instantiated*, and the resulting *constituent structure* substituted into the tree in place of the original elmt-instance. This process will be described in section instantiating_phrases_n.y.w.. Constituent structure is discussed in the next section.

PHRASES do not have properties *per se*. They are expressions, and have a structure defined by the grammar in the accompanying figure. @@.

6.2 The Dictionary

The job of the linguistics component is to decide what text to select for a given message. This decision can be broken down into a large number of smaller decisions, each of which is an instance of a "generic" decision—part of the component's long term knowledge about production. That knowledge can be loosely divided into three kinds.

(1) knowledge of use: how can objects be described in a natural language; what are the criterion that determine the choice of wording, syntactic organization, and degree of specificity and detail for a given object in a given pragmatic and intentional context?

(2) knowledge of control: what are the dependencies between the decisions involved in realizing a message; in what order should they be made if the evaluation sequence is to be optimal, i.e. one that requires no wasted effort?

(3) knowledge of grammar: what are the legitimate patterns of linguistic relations among the words of a text; what is the form required of a specific text if it is to express specific linguistic relations in a specific linguistic context?

Virtually all of the linguistic component's knowledge of the first kind is contained in its *dictionary*, the subject of this section. (The others are covered in the two following sections.) This section will discuss how "usage-knowledge" is packaged within the dictionary and how it is manipulated by the realization procedure. Later, in chapter four, several notational elaborations

of the dictionary's structure will be introduced to permit the knowledge to be conveniently shared, and in chapter three, a technique for using the dictionary as the basis of predictive symbolic reasoning will be described.

The structure of the dictionary

The dictionary consists entirely of individual entrys, one for each distinct object or object class which may occur in a message. In programming terms, it is a heap: a random-access memory store. Its internal structure—the "glue" holding together the entrys—needs to be no more extensive than is required to implement the interface function entry-for.

First the relationship of entrys to the domains for which they are written will be discussed. Then the data-types of the dictionary: ENTRY's DECISIONS, DECISION-RULES, and CHOICES, are described. The "load-time syntax" for these types as currently used in MUMBLE will be included to provide a convenient way to present entries in the chapters to follow. Several examples of entries will then be given and walked through before moving on to the discussion of the realization procedure.

Computational environment

Conceptually, entrys are extensions of the speaker that have been embedded within the linguistic component. They straddle the line between the two decision-making domains, using pragmatic criteria to decide among linguistic options. However, it is the linguistic component that controls when an entry will be used, and consequently provisions must be made (1) in the entrys themselves to provide a representation of their decision procedure that the linguistics component can manipulate, and (2) in the realization procedure to incorporate an evaluator which is matched to the representation the speaker employs and has access to the speaker's environment.

Within entrys, expressions that test the state of either of the decision-making domains will be objects of type PREDICATE. Because of their potential breadth, the functionality of these expressions (when evaluated) can only be given indirectly. That is, their domain is those objects that are the values of accessible local-variables, controller-variables, or ACCESS-EXPRESSIONS (i.e. all of their syntactically possible arguments), and their range is BOOLEANS. Beyond their functionality, of course, nothing can be said about them except in the context of a specific speaker and expert program.

In MUMBLE, the two provisions have been easily arranged, since both the linguistic component and all of the micro-speakers have been written in a common programming language, LISP. Entry s) incorporate code fragments embodying the test and data access they required, and the realization-procedure uses the standard LISP function eval. To provide access to necessary data in the speaker, the full system either runs in a common address space, or this is simulated (see section <computational_environment>).

Decomposing message elements

It is in the entrys that the decomposition of composite msg-elmts takes place. Any entry that does this must be written so as to contain or to be able to access the necessary decomposition procedures, employing the customary expressions of the domain to do so as discussed earlier (pg.50) Once the subelements have been obtained, they will typically need to be tested for various linguistic and domain-specific properties and will ultimately be positioned in the tree by a CHOICE. This necessitates some means of refering to them while they are being manipulated. In MUMBLE, this is done through the use of local-variables. When an entry is interpreted for some ELMT-INSTANCE, that ELMT-INSTANCE is made the value of the ENTRY's input parameter where it can be used as an argument to decomposition functions in the defining expressions of the other local-variables. Such expressions will be objects of data-type ACCESS-EXPRESSION; their input domain is the same as for PREDICATEs and their output is a MSG-ELMT. Once given values (through the evaluation of their corresponding ACCESS-EXPRESSIONS) local-variables can then be used as arguments to the various testing PREDICATEs in the DECISION-RULEs and ultimately the selected CHOICE.

Data type: ENTRY

The ENTRY for a msg-elmt describes what the possible realizations of that element are and under what conditions each of those possibilities is to be selected. Apart from the information extracted by the interface functions for use in pronominalization, these are the only properties of MSG-ELMTs that are of interest to the linguistics component. In particular, any semantic or pragmatic generalizations among the elements will not be noticed except insofar as they are compiled into the structure of the dictionary.

A single entry may serve an arbitrary number of distinct MSG-ELMTs; indeed, in the experience with MUMBLE, this has been the usual case—each entry is designed to serve a natural class within the system's domain, e.g. "logical implications" or "tic-tac-toe moves". This is arranged by having the function entry-for so designed that all elements in that class are assigned to the same entry, and making the particular msg-elmt involved in each case accessible within the entry via a pre-determined parameter.

Like the dictionary, each entry is largely only a shell—a placeholder where other objects can be put. The structure can be summarized as follows:

> An ENTRY consists of one or more DECISIONS. Each DECISION consists of one or more DECISION-RULEs, which, in turn, consist of zero or more PREDICATEs and a CHOICE.

The point of this multi-level design for ENTRY's is to make them easier to inspect symbolically and to bring them under contextual control. It is a formalism for encoding a decision-procedure, specifically a discrimination network, as a declarative structure that can be readily analysed by special purpose predicates as well as be interpreted or compiled into an

efficiently executing procedure. The compiler will not be discussed; the interpreter is the realization procedure to be discussed shortly (itself a collection of interpreters for the various data-types); and the special purpose predicates will be discussed in section <analysis_routines>.

The entry itself has the following properties:

Decisions-to-make An ordered list of one or more DECISIONs. An ENTRY's decisions are evaluated sequentially, following the order given by this list. The matrix decision is always first.

Parameters A list of one or more LOCAL-VARIABLEs, bound whenever the entry is interpreted to the input parameters of the entry as determined by interface function entry-arguments-for.

Entry-level-variables Optional. A VARIABLE-DEFINITION-TABLE (a list of LOCAL-VARIABLE—ACCESS-EXPRESSION pairs) used for the decomposition of message elements.

mode A symbol: either *unit* or *sequence* Used by the entry-interpreter to determine how to combine the CHOICEs of the ENTRY's DECISIONs. If omitted from an ENTRY's specification, *unit* is assumed.

will-be or could-be Either a list of constituent-structure-labels that describe the possible-realization that the entry will select, or a list of such lists, describing the selections that it could make.

Data type: DECISION

DECISIONs are a mechanism for facilitating the introduction of modularity and useful generalizations into the decision-making process. In some strict sense, they are unnecessary, since entries designed using multiple decisions can always be "straightened out" into one large decision by multiplying out their choices and conditions However, it has been my experience with the micro-speakers that the process of analysing message elements and constructing realizations decomposes naturally into sets of loosely-coupled decisions.

Individual DECISIONs are made up of the following properties:

decision-name A NAME. DECISIONs are typically not independent objects but are defined relative to a particular ENTRY. Shared Decision-names often have a functional significance: DECISIONs with the same name in different ENTRYs will perform comparable parts of the work. Those DECISIONs defined independently (either because the identical one will be used in many ENTRYs or because it is a GRAMMATICAL-DECISION (see grammatical_decisions_n.y.w.), will have names (as all objects do), but will not have decision-names unless actually associated with an ENTRY.

gating-condition A PREDICATE. If it is true, the decision will be made, otherwise the entry-interpreter will skip over it. If omitted, the DECISION is assumed to always be applicable.

decision-rules A list of DECISION-RULEs.

filters A list of CHOICE-FILTERS.

default A CHOICE-APPLICATION. Optional unless no DECISION-RULEs are specified.

decision-level-variables An optional VARIABLE-DEFINITION-TABLE

The relationship of decisions to entry modes

The interpretion of an entry results in the construction of a single object of type possible-realization. This is even though the process may involve the making of an arbitrary number of decisions—each with its own choice—according to the value of the entry's decisions-to-make property. Obviously the construction efforts of the individual decisions must be merged. Two kinds of merging have proved useful: (1) having every decision add to a common "matrix"—this the mode *unit*, and (2) interpreting the decisions as adding sucessive items to a sequence—the mode *sequence*. In both cases, the entry must have a decision with the decision-name matrix, whose job it is to select the common object to which the other decisions will refer. Within *unit* entrys, this object is accessible to the other decisions via the local-variable matrix (bound by the entry-interpreter), whereas within *sequence* entrys, no explicit references need to be made because the construction of the sequence is done behind the scenes by the entry-interpreter. The matrix decision of a *sequence* entry will always select a phrase involving a constituent-schema that may have an arbitrary number of constituents (pg.<schema_with_arbitrary_numbers_of_constituents_n.y.w.>).

Data type: DECISION-RULE

conditions A list of zero or more PREDICATES that the decision-rule-interpreter is able to evaluate.

choice A CHOICE-APPLICATION.

A DECISION-RULE is a specification of the conditions under which its choice may be selected by the DECISION(s) of which the DECISION-RULE is a member, namely only if *all* of the conditions of the DECISION-RULE are satisfied. Note that this is a specification of necessary conditions, not sufficient ones. The relative order of the DECISION-RULE relative to the others in the DECISION will ultimately determine which otherwise plausible CHOICE is selected.

Data type: CHOICE

Choices are the specifications of POSSIBLE-REALIZATIONS. They play several roles: expressing generalizations about usage, organizing and simplifying the decision-making process, and abstracting that process away from the largely irrelevant details of constructing the realizations themselves. CHOICEs are parameterized objects, each potentially shared by many ENTRY's. (A CHOICE may even be repeated several times within a single ENTRY if different CONDITION-SETs select the same one.)

What actually appears in the entries is not a CHOICE but a CHOICE-APPLICATION—an expression with the following structure:

CHOICE-APPLICATION ::= (<operator> <argument>*)
<operator> ::= { **choice** | **decision-extension**[25] }
<argument> ::= { LOCAL-VARIABLE | ACCESS-EXPRESSION }

A CHOICE-APPLICATION is interpreted by the choice-interpreter, which determines the values of its arguments and applies them to the CHOICE. Thus the domain of CHOICEs is the set of objects which can be the values of current LOCAL-VARIABLES or which can be computed by the available ACCESS-EXPRESSIONS. Loosely speaking, their range is single POSSIBLE-REALIZATIONS. (In mult-decision entrys, most of the choices are interpreted for their side-effects on the matrix.) It follows that any ENTRY that has a decision with more than one choice constitutes a non-deterministic function from one msg-elmt (the one being realized) to one of several POSSIBLE-REALIZATIONS. As choices may be shared by an arbitrary number of entrys, neither of these functions are bijections, i.e. the same POSSIBLE-REALIZATION may be selected as the realization of more than one msg-elmt.

All choices have the following properties:

choice-will-be A list of one or more symbols describing the realization to which the choice corresponds.

parameters A list of zero or more LOCAL-VARIABLES, used by the other properties of the choice to stand for instances of message elements. The choice-interpreter will assign values to these parameters in accordance with the order of message elements specified by particular CHOICE-APPLICATIONS.

---

25. In the spirit of programming clarity, a designer may write CHOICE-APPLICATIONs whose operators, instead of being CHOICEs, are the names of entire ENTRYs or decisions (lumped together as the syntactic type decision-extension). In such cases, what would otherwise be PREDICATEs duplicated by many DECISION-RULEs have been grouped into one, with the varying predicates (and the choices they eventually pick out) moved into their own unit, that may now be used as a *subroutine*. In effect, instead of selecting a unit action, the DECISION is selecting a whole new set of DECISIONs. This technique is also useful in designing the DISCOURSE-HISTORY, since the "choice" that is recorded for a DECISION can be made to be at a more appropriate level of abstraction. For elaboration and examples see section <decision_extensions>.

marking-actions A list of zero or more MARKING-ACTIONS (expressions involving operators that add attachments to an elmt-instance or add new or impure DECISIONS). These will be evaluated for their side-effects on ELMT-INSTANCES being embedded in the phrase.

mode A SYMBOL: either *matrix* or *refiner*. The mode property indicates whether the choice will cause a new object to be contructed or will instead refine a specific existing one.

Choices that construct possible-realizations

A DECISIONS with the decision-name *matrix* (of which there is one per entry) and any DECISIONS in an ENTRY whose mode is sequence will employ CHOICES whose mode is *matrix*. Such CHOICES have special properties according to the type of POSSIBLE-REALIZATION that they construct. The choice-interpreter distinguishes between them on the basis of which of these properties they have.

Most *matrix* CHOICES construct an instance of a PHRASE (i.e. a NODE) for which purpose they have these properties:

phrase A PHRASE.

map A map—an ASSOCIATION-LIST pairing parameters of the CHOICE with SLOT-PATHS.

associated-transformations A list of TRANSFORMATIONS. See II.4.5.

CHOICES that result in ELMT-INSTANCES are quite simple. The source MSG-ELMT is given as an argument in the CHOICE-APPLICATION and identified within the CHOICE as the value of the LOCAL-VARIABLE specified by the property:

element-returned A LOCAL-VARIABLE.

CHOICES that result in TRACES are treated as special cases, and specific, constant CHOICES are used, i.e. *null_word*, *null_utterance*, *np_trace*, and *ellipsed_verbal*. (The names of CHOICES are, by convention, chosen to be short descriptions of the realization the specify. Nothing anywhere in the linguistic component will react to these symbols except to compare them with other names, thus the choice of wording for a name is totally at the convenience of the designer.) Each trace-leaving choice takes a single argument: the object to be realized by the TRACE. The reason for four CHOICES where one would suffice logically is to provide an annotation of the reason why the selection was made; see <choices_as_annotation>.

Choices that refine possible-realizations

As their name implies, *refiner* CHOICEs act upon POSSIBLE-REALIZATIONs that have already been constructed by a *matrix* CHOICE. They may add properties, fill empty SLOTs, mark argument elmt-instaces, or make certain non-destructive modifications to constituent structure (see pg.44). They may *not* delete or change any already established properties. The only linguistic object that these CHOICEs may refer to is the one created by the *matrix* DECISION of their ENTRY (and possibly augmented by later DECISIONs). It is accessible through the LOCAL-VARIABLE matrix.

At this writing, *refiner* CHOICEs in MUMBLE are relatively unstructured. They have one property:

Actions A list of expressions each of which, when evaluated, will add to the properties of an object indicated by one of their arguments.

This (admittedly vague) specification acts to rule out the use of arbitrary programs. All expressions must employ property adding operators directly, making it feasible to automatically analyze the effects of a *refiner* CHOICE, given a separate annotation of the effect of each of the operators it employs.

Data type: CHOICE-FILTER

conditions A list of PREDICATEs.

choices-to-be-removed Either an explicit list of CHOICE names or a PREDICATE testing properties of PHRASEs.

CHOICE-FILTERs are a means of controlling entire systems of CHOICEs at once on the basis of their properties. They are associated in the grammar with certain CATEGORYs and SLOT-NAMEs and may be included in DECISIONs. An example of a chice-filter appears on page 31.

vi. The load-time syntax of the dictionary

The accompanying figure gives the grammar of the load-time syntax currently used in MUMBLE to specify ENTRYS, DECISIONS, DECISION-RULE and CHOICEs. It is summarized as follows. All defining expressions are fully parenthesized, i.e. they are legitimate LISP s-expressions. The first token is always the name of an object-creating macro. The second token is always the object's name; if the object has a parameters property its value will be on the same line. The object's other properties follow, paired with a key-word.

THE LOAD-TIME SYNTAX OF THE DICTIONARY

```
ENTRY ::= (define- entry NAME <parameter-list>
                    {variables VARIABLE-DEFINITION-TABLE}
                    { mode {unit|sequence}}
          <decision-form>*)
<parameter-list> ::= (LOCAL-VARIABLE*)
 VARIABLE-DEFINITION-TABLE ::= (<variable-definition>*)
 <variable-definition> ::= (LOCAL-VARIABLE ACCESS-EXPRESSION)
 <decision-form> ::= (decision-name{NAME|<decision-body>})
 DECISION ::= (define-decision NAME <decision-form>)
 <decision-body> ::= (define-decision NAME
                    {variables VARIABLE-DEFINITION-BLOCK}
                    {gating-condition PREDICATE }
                    {default CHOICE-APPLICATION}
                    DECISION-RULE*)
 DECISION-RULE ::= (PREDICATE* CHOICE-APPLICATION )
 choice ::= (define-choice name <parameter-list>
        {mode{matrix|refiner}
        {{{phrase phrase}
          {map map}}
         |{element-returned local-variable}
         |{actions(expression*)}}
        {marking-actions (marking-action*)}
        )
```

vii. Stepping through an example

At this point, it will be useful to put some flesh on these definitions by looking at some actual dictionary entrys. The ENTRY below, *implication_entry*, was developed for the logic domain. As its name suggests, it is the ENTRY for the logical connective "—".

```
(define-entry implication_entry (wff)
    variables ((antec (antecedant wff))
               (conseq (consequent wff)))
    (matrix
        default (if-then antec conseq)
        ((A-is-a-subject-for-predicate-B antec conseq)
         (subj-pred antec conseq))
        ((propositionp antec)
         (implies antec conseq)) ))
```

Implication_entry has two LOCAL-VARIABLES: *antec* and *conseq*, and has one DECISION: *matrix*, with a default and two DECISION-RULES, each with one predicate

Let us suppose that the ELMT-INSTANCE to be realized was an instance of the time-worn proposition.

man(socrates) — mortal(socrates)

The first thing that would happen is evaluation of the VARIABLE-DEFINITION-TABLE to bind the two LOCAL-VARIABLES. The value of *antec* would become man(socrates), and of *conseq* mortal(socrates). Then the DECISION is looked at: There is no specified gating-condition, therefore by default the decision is to be made, and we go on to evaluate its DECISION-RULES in order.

The first DECISION-RULE (and the second as well) has only one predicate. *A-is-a-subject-for-predicate-B* is described in detail on page <a_is_a_subject_for_predicate_b_n.y.w.>. It is a general predicate that examines the ENTRYs of its two arguments to see if is possible to realize the first as the [subject] of the second as in "*all men are mortal*". In this case the antec(dent) can only be expressed as a clause and the predicate fails. Going on to the second DECISION-RULE, it too fails because of the particular way "proposition" is defined in the logic domain.

This means that the default is taken. Its CHOICE-APPLICATION consists of the CHOICE *if-then* applied to the two subelements picked out by the LOCAL-VARIABLES. Carrying through this application results in the construction of the complex clause below.

clause

[if-slot] [then-slot]
man(socrates) mortal(socrates)

A more complex entry

The ENTRY below, *concept-defining_entry*, is typical of the more complex entrys that have been written for the micro-speakers. It is from the KLONE-nets-as-objects domain—each of the paragraphs of the sample texts on page <klone_sample_n.y.w.> were constructed by it. It is a discourse ENTRY, organized around the internal structure of the message elements it realizes (KLONE "concepts"): each of its three non-trivial DECISIONs is responsible for the realization of a different kind of subelement. (The pattern-matching syntax used here is explained on pg.<role_entry_n.y.w.>.)

```
(define-entry concept-defining_entry (concept)
    mode sequence
    (matrix
        default (discourse-with-focus concept) )
    (describe_super-concepts
        variables ((superc-relation (collect !(subconcept ,concept ??))))
        default (say-fact superc-relation)
        ((given superc-relation)
         (say-nothing)))
    (describe_roles
        variables ((roles nil)  ;set by side-effect of next access-expression
                   (role-relations (collect !(has-role ,concept >roles))) )
        default (give_the_restrictions_with_the_roles roles role-relations)


    (describe_any_subconcepts
        condition (collect !(subconcept ?? ,concept))
        variables ((daughters nil)
                   (subc-relations (collect !(subconcept >daughters ,concept))) )
        default (push_subconcepts_no_summary daughters)
        ((all-trivial-concepts daughters)
         (summarize_don't_push_subconcepts subc-relations)) ))
```

6.3 The Realization Procedure

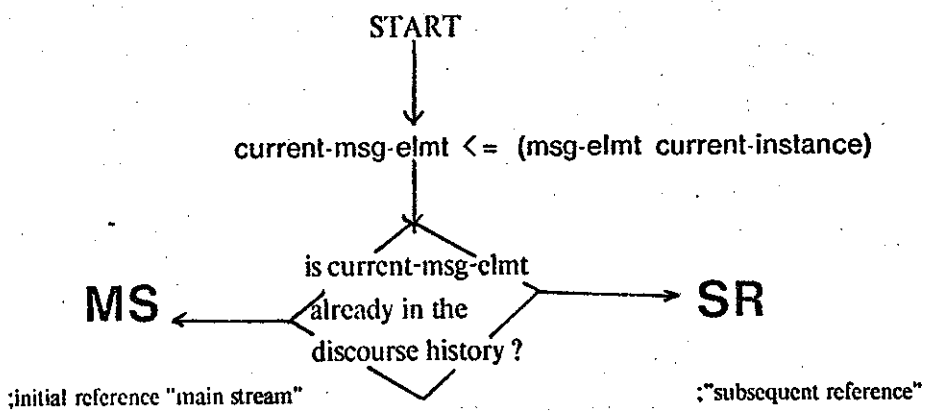
The realization procedure takes a single ELMT-INSTANCE as input and selects a single POSSIBLE-REALIZATION as its output. It is itself embedded within the controller, which determines what ELMT-INSTANCEs the realization procedure will receive and where the POSSIBLE-REALIZATIONs it produces will be embedded.

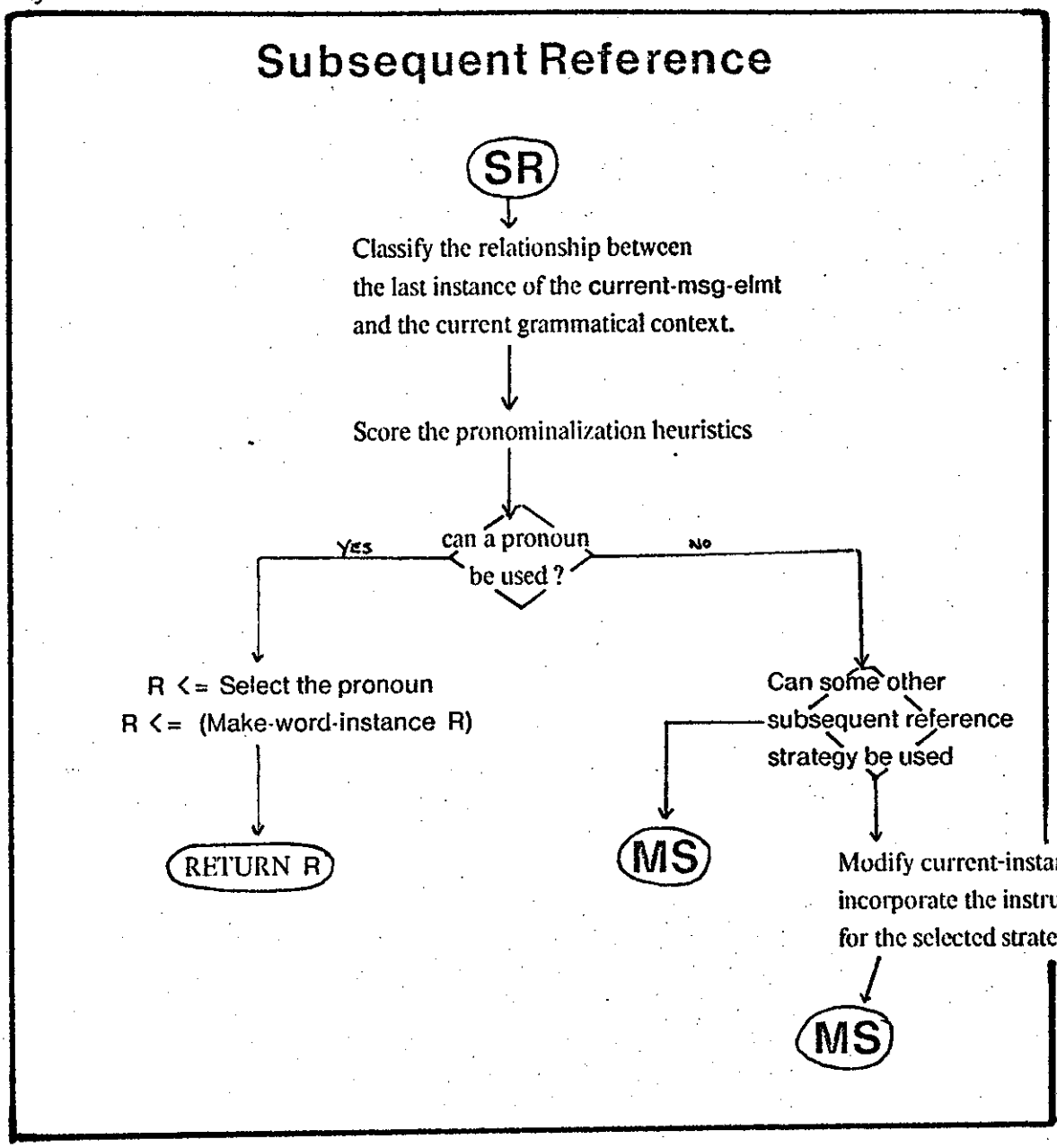
Conceptually, all of the information needed to decide upon the realization should be found in the entry for the elmt-instance. However, in actual practice, it is expedient to extract material that would otherwise be common to every entry and install it as sharable modules. This has been done for subsequent reference and for syntactic transformations. The result is shown in this flowchart.

The Realization Procedure (current-instance)

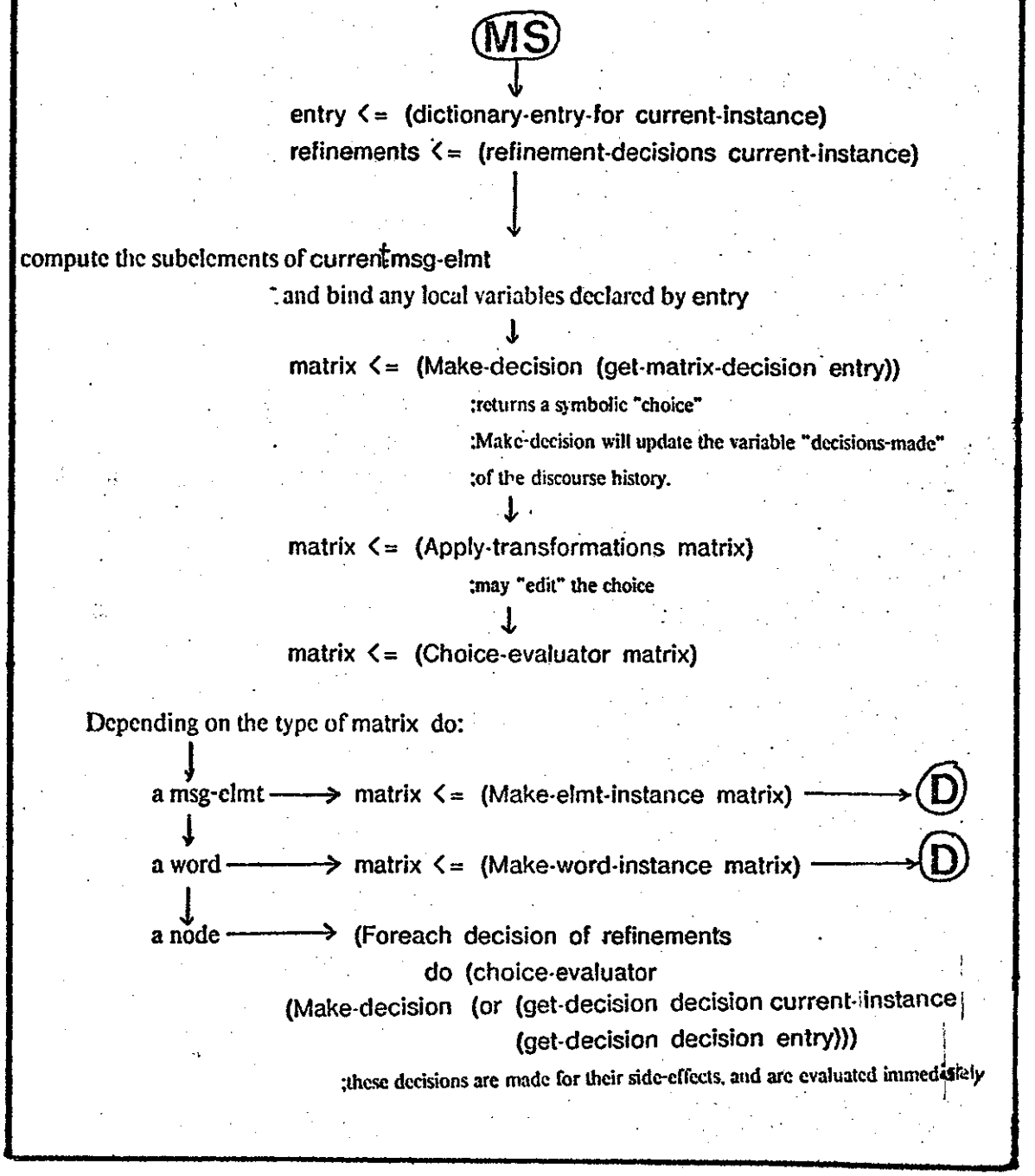
argument: an elmt-instance

return value: a word-instance, a node, or a new elmt-instance




Subsequent Reference
SR
Classify the relationship between
the last instance of the current-msg-elmt
and the current grammatical context.
Score the pronominalization heuristics
can a pronoun
be used ?
Yes
No
R <= Select the pronoun
R <= (Make-word-instance R)
RETURN R
Can some other
subsequent reference
strategy be used
MS
Modify current-instance to
incorporate the instructions
for the selected strategy
MS

Main Stream Realization



(Foreach decision of (default-decisions (category matrix))
do (or (member decision decisions-made)
(make-decision decision matrix)))

↓

Unbind any local variables defined during the processing of
the entry and any decisions

↓

RETURN matrix

Make-decision (D)

argument: a decision
return value: a choice-application

↓

bind any local variables declared by D

↓

choice-set <= (choices D)

↓

choice-set <= (Apply-filters current-grammatical-filters choice-set)
;Any choice for which one of the filters evaluates to "true" is
;omitted from the new choice-set.

↓

(Foreach decision-rule of D whose choice has not been filtered out
do 1. (Foreach predicate of the condition-set of the decision-rule
do 1. evaluate the predicate
2. if it is satisfied, continue;
if it is not satisfied, escape from this loop

If every predicate in the decision-rule was satisfied,
RETURN (choice decision-rule)

If no decision-rule has succeeded, then either

a. (default D) is defined in which case
RETURN (choice (default D))
b. or ERROR

As you can see, subsequent reference (i.e. textual references to an object after it has been introduced into the discourse) is a separate path through the procedure, completely divergent for pronominalization but eventually merging with the "main stream" for the other forms. I will postpone any discussion of this pathway until the next chapter (VII.B.1.1).

The realization procedure is made up of a set of independent interpreters, following the decomposition of usage-knowledge into the different data-types of the dictionary. Their functionality is given below.

entry-interpreter ELMT-INSTANCE to POSSIBLE-REALIZATION
decision-interpreter DECISION to CHOICE-APPLICATION
decision-rule-interpreter DECISION-RULE to BOOLEAN
transformation-interpreter CHOICE-APPLICATION to CHOICE-APPLICATION
choice-evaluator CHOICE-APPLICATION to POSSIBLE-REALIZATION

What follows is the description of the "normal case" procedure. Deviations caused by marked ELMT-INSTANCEs are discussed at the end in section <deviations_from_the_'normal'_procedure>.

i. The Entry-Interpreter

The entry-interpreter plays a managerial role, coordinating the actions of the other parts of the realization procedure. It itself is responsible for the following tasks:

(1) binding the ENTRY's parameters to the MSG-ELMTs specified by the entry-arguments-for property of the ELMT-INSTANCE;

(2) computing any needed subelements of the ELMT-INSTANCE by evaluating the variable-definition-table if there is one.

(3) applying the decision-interpreter to each of the decisions on the decisions-to-make property of the elmt-instance.

(4) Applying the transformation-interperter to each choice-application that involves a choice whose mode is *matrix*.

(5) applying the choice-evaluator to the choice-applications the decision-interpreter returns and combining the results according to the mode of the entry. These actions are done in line—the selection of the previous decision is completely processed before the next decision is begun.

(6) binding the local-variable matrix to the possible-realization constructed by processing the selection of the matrix decision.

(7) making any applicable DEFAULT-DECISIONs that have not been overridden by

specific DECISIONs already made (see pg.<default_decisions>);

(8) having a record made of the results of this effort: decision-name—name of the CHOICE made for them pairs. (These names implicitly include a record of any transformations that were applied).

Energizing variables

As we are dealing here with an "interpreter", explicit provisions must be made to insure that any variables defined by an ENTRY are given their appropriate values by the time that they referenced in any expressions. Using the VARIABLE-DEFINITION-TABLE as input, this operation takes each of its LOCAL-VARIABLEs in turn and binds it to the value returned by evaluating the ACCESS-FUNCTION with which it is paired in the table. This binding remains in effect until the interpretation of the ENTRY is completed. local-variables are defined and bound in the order that they appear in the table; later access-functions may refer to the values of earlier local-variables.

Some predicates that text properties of msg-elmts will need to reference the current elmt-instance associated with the msg-elmt rather than the msg-elmt itself. Cases where this is needed can be detected at system load-time by the postprocessor and their expessions are edited slightly to provide instances rather than elements when finally evaluated.

ACCESS-FUNCTIONs and non-linguistic PREDICATEs are evaluated with the domain-evaluator—part of the interface that must potentially be redesigned for each new domain. Unlike the other parts of the realization procedure (which understand the objects in the dictionary only syntactically) this evaluator must have access to their computational meanings as well, i.e. access to the computational state of the speaker and expert program which the expressions will probe.

ii. The decision-interpreter

The decision-interpreter first checks that the DECISION is applicable by evaluating its gating-condition. If it is not, the interpreter immediately returns a "dummy" CHOICE-APPLICATION named *decision-not-applicable* for the benefit of record keeping.

If the DECISION is applicable, it is filtered (see below) and then each of its remaining DECISION-RULEs in turn is passed to the condition-set-interpreter until one of them is satisfied. The choice of the satisfied DECISION-RULE is then returned. If none of the DECISION-RULEs are satisfied, the DECISION's default is returned. If none are satisfied and there is no default, then there has been an error in design.

Applying CHOICE-FILTERs CHOICE-FILTERs come from two sources: the grammar and the DECISION itself. (The set grammatical filters that applies will change from moment to moment as the grammatical context changes. Consequently they are accessed through the controller-variable, *grammatical-conditions-on-choices*, whose value is a list of CHOICE-FILTERs.) Each filter is tested for applicability by evaluating its predicates in the same way one evaluates the predicates of

DECISION-RULEs (i.e. all of them must be satisfied if the filter is to be applied). The choices-to-be-removed of applicable filters are then compared with the set of possible CHOICEs of the DECISION, either by a literal match against names or in terms of a PREDICATE testing properties of the phrase of the CHOICE.

All DECISION-RULEs that select CHOICEs that meet the filter are then removed from the set of "viable" DECISION-RULEs that can be used as the basis of the decision. (In MUMBLE, its job is made easier by an extensive body of cross-indexes compiled at system load time. Trading off space for time, these provide a direct link from each maximally transformed CHOICE to the DECISION-RULEs that would select it.)

If all of a DECISION's CHOICEs are filtered out, there has been a design error somewhere "upstream" in the decision-making process—the message element that now cannot be realized should never have been put in this context. Analysis routines to help avoid this condition are discussed in section <analysis_routines>.

iii. The decision-rule-interpreter

The action of this interpreter could be described in LISP terms as applying the function AND to the list of conditions and returning the result—*true*, if each of the PREDICATEs in the list evaluated to *true*, otherwise *false*. The PREDICATEs are evaluated in the order that they appear in the list.

iv. The transformation-interpreter

TRANSFORMATIONs (see II.4.5) are functions from CHOICE-APPLICATIONs to CHOICE-APPLICATIONs, their output CHOICE-APPLICATIONs being applications of a *new* CHOICE to the *same* arguments as appeared in the old choice-application. Each CHOICE has a transformations property, that is effectively a list of all of the TRANSFORMATIONs that could ever apply to it on the basis of patterns of SLOT-NAMEs and CATEGORYs within those PHRASEs, e.g. [subject] with [object-1] for active to passive, or a noun phrase with [of] for genitive to possessive. In MUMBLE, this list is compiled at system-load time by a postprocessor.

Each TRANSFORMATION has an associated list of CONDITION-SETs, that define the grammatical, thematic, and discourse contexts in which it applies. The interpreter evaluates these lists as though they were part of a DECISION, (which, in effect, they are—decisions whether to use a given transformation or not). If one of the DECISION-RULEs is satisfied, the interpreter applies the TRANSFORMATION. There is no limit to the number of TRANSFORMATIONs may be applied.

Transformations selective modify CHOICEs. They are defined in terms of copying and editing operations that take the phrase and map of the CHOICE to be transformed and produce a new phrase and map, identical to the originals except for local alterations specified by the transformation. (For details, see <applying_transformations>.) Note that these "transformations"

act upon specifications for constituent structure rather than upon constituent structure itself, as they take place before the specification is instantiated.

When a TRANSFORMATION is applied, the new CHOICE that results becomes the basis of the list of (further) TRANSFORMATIONS to be tested. The reason for this can be shown by this hypothetical example: Suppose the "basic" CHOICE for a certain message element (the one given in the ENTRY before any TRANSFORMATIONS have been applied) calls for using the English verb *expect* e.g. say the message element could be modeled as:

(*expect* <somebody> <do_something>))

Suppose that the identity of the <somebody> is not relevant and that consequently the first TRANSFORMATION to be applied is *passive*. If we stopped there, we might get the text:

"*<something being done> was expected.*"

However, the grammatical relations in that text have now assumed a pattern that would permit us to now also apply the (sometimes stylistic) TRANSFORMATION *extraposition* to get:

"*It was expected that <something was done>.*"

If the set of applicable TRANSFORMATION did not change as the choice changed, possibilities like this would constantly be missed.

v. Choice-evaluator

The choice-evaluator is partly an applicative order evaluator and partly an interpreter of CHOICES. The first thing it does is to process the arguments in the CHOICE-APPLICATION. LOCAL-VARIABLES are replaced by the elmt-instances that are their current values; ACCESS-EXPRESSIONS are evaluated with the domain-evaluator to yield MSG-ELMTS and the MSG-ELMTS are then replaced by ELMT-INSTANCES. These are then paired with their sequence order counter-parts in the parameters property of the CHOICE. (Note, the argument expressions should *not* have side-effects. Furthermore, neither how the choice is interpreted nor the form of the output may be conditional on specific arguments except trivially in that those are the arguments mapped into the output.)

From this point, how the CHOICE will be interpreted depends upon its properties. If it is to return a TRACE, then the link property of a new TRACE of the appropriate sort is set to the object which is the (sole) argument to the CHOICE-APPLICATION. CHOICES with element-returned properties will return the ELMT-INSTANCE that is paired with the LOCAL-VARIABLE that is the value of that property, and any marking-actions there may be are evaluated.

If a CHOICE has a PHRASE, first that PHRASE is instantiated, creating a structure of NODES, and then the MAP is interpreted. Each ELMT-INSTANCE now paired with a parameter in the MAP becomes the contents of a SLOT in the instantiated PHRASE according to the directions given in the SLOT-PATHS associated with the parameter in the MAP. Finally, the marking-actions, if any, are

evaluated, adding attachments or IMPURE-DECISIONS to the newly embedded ELMT-INSTANCES. The instantiated phrase is what is returned.

For example, below is the CHOICE written for the Macbeth domain to encode the "sentence builder"[26] idiom "X *is a character in* Y".[27]

```
(define-choice part-of_choice (value)
    phrase (vp_predicate-nominative ()
                pred-nom (regular-np ()
                            head "character"
                            qualifiers (regular-prepp ()
                                        prep in )))  )
                map ((value . (pred-nom qualifiers prep-obj)))  )
```

To evaluate this CHOICE, we first instantiate the PHRASE *is-a-character-in*, yielding a newly created constituent structure identical in pattern to the one on page <is_a_character_in_constituent_structure_n.y.w.>. To interpret part-of_choice's map, the value of the local-variable in the first (and only) argument position of the CHOICE-APPLICATION being evaluated is looked up and a ELMT-INSTANCE created for it if there is not one already. That elmt-instance is then positioned in the constituent structure as the contents of the SLOT indicated by the SLOT-PATH (pred-nom qualifiers prep-obj). This involves using the SLOT-PATH to pick out successive SLOTs in the immediate constituents of each successively further embedded NODE starting from the root NODE (i.e. the verb phrase).

This CHOICE has no marking-actions (but see the CHOICE on page <universal_instantiation_choice_n.y.w.>). Consequently the evaluation process is finished and the newly constructed constituent structure is returned.

vi. Default decisions

Like TRANSFORMATIONS, DEFAULT-DECISIONS are another facility for representing generalizations in the decision-making process. They are intended to deal with details or locally unpredictable events that individual ENTRYS should not be burdened with. Examples include: setting the *relative-head* hook on noun phrases with relative clauses—a detail; and reacting to attachments calling for intensifying a NP with a reflexive ("*Giuseppi himself*") or for marking polar contrast in a clause ("*Some barber* does *shave everyone who...*")—both unpredictable events.

26. A term due to Becker from his insightful paper *The Phrasal Lexicon* [becker_phrasal_lexicon].
27. The CHOICE specifies a verb phrase (rather than the full clause that the idiom appears to call for) because of the way the *default-framelet-entry* (page <default_framelet-entry>) is designed to function. This CHOICE will be used as the realization of the "property", (part-of MA), rather than the whole framelet, (macbeth (part-of MA)).

Conceptually, default-decisions are another variety of GRAMMAR-ROUTINE—a procedure associated with a constituent-structure-label. But because their triggering conditions are so different from the other GRAMMAR-ROUTINES, they have their own type. (See section <attachments_and_default_decisions> for examples and a discussion of the operations they can employ.)

DEFAULT-DECISIONS fall into classes according to the matrix CHOICE of the ENTRY. MUMBLE, for example, has three classes: those for when a clause is chosen, those for a noun phrase, and those for an ELMT-INSTANCE. The class of a DEFAULT-DECISION is given by the property:

> Relevant-when . A PREDICATE testing some property of the CHOICE made by the matrix DECISION. The predicate must be satisfied if the DEFAULT-DECISION is to be made.

The other properties of DEFAULT-DECISIONS are the same as those of "regular" DECISIONS.

The entry-interpreter makes DEFAULT-DECISIONS after it has made all of the regular DECISIONS (recall the diagram on page <realization_procedure_diagram_n.y.w.>). If any of the regular DECISIONS have the same decision-name as one of the DEFAULT-DECISIONS, the regular DECISION takes precedence and the default is not made—hence the name.

vii. Recording the results

In natural language production, a critically important decision-making resource is a record of past decisions. The entry-enterpreter adds to this record every time an ELMT-INSTANCE is realized as a linguistic object. While DECISIONS are being made for a given elmt-instance, the interpreter compiles an ASSOCIATION-LIST, pairing the decision-name of each DECISION with the name of the CHOICE that it made on this occasion. When the elmt-instance is finally realized, this list is passed as an argument to a HISTORY-TAKER (history_taker_n.y.w.), where it becomes part of the RECORD that is compiled for the ELMT-INSTANCE's msg-elmt in the long term discourse-history.

viii. Deviations from the 'normal' procedure

In writing a dictionary, a profitable approach is to begin by developing ENTRYs for the natural classes of objects in the domain, and then to move on to those objects which, because of intentional or discourse context, are similar to the objects in the natural classes "but not quite". The new-decisions and amended-decisions properties on ELMT-INSTANCES are a mechanism for dealing with such objects. MARKING-ACTIONS in CHOICES are used to set these properties on ELMT-INSTANCES as the instances are being embedded in the realization. Later, the realization of their will be done partly by DECISIONS from the "normal" ENTRY associated with them by

entry-for, and partly by new DECISIONs or augmented IMPURE-DECISIONs attached to them by the CHOICEs.

New-decisions are made after the "normal" DECISIONs and before any DEFAULT-DECISIONs, and therefore may override defaults. Amended-decisions are instructions for modifying or preempting certain of the normal DECISIONs. They are organized as objects of type IMPURE-DECISIONs with the following two properties:[28]

Decision-name Must correspond with the decision-name of one of the normal DECISIONs.

Preempting-choice A CHOICE-APPLICATION.

IMPURE-DECISIONs are looked for by the decision-interpreter on the basis of common decision-names. If one if found and there is a preempting-choice, then none of the DECISION-RULEs of the corresponding normal DECISION are tested and the CHOICE-APPLICATION of the preempting-choice returned instead. Preempted-choices are a device for avoiding redundant effort when some later DECISION is a predictable consequence of an earlier one. See for example <emphasize_polarity>.

28. In earlier versions of this theory, there were also properties for new decision-rules to be added to existing decisions at various places with respect to those already there. However, the "meta-decision" facility described in section meta_decisions_n.y.w. proved to be a clearer way to accomplish the same goals.

CHAPTER THREE

SOME ENGLISH CONSTRUCTIONS

As any reader of the last chapter will attest, the linguistic component is complex. It employs twenty six different data types, builds a full-scale tree of linguistic relations over the texts it produces, and independently records the history of each of the objects it realizes and the grammar decisions it makes. For an investment of this size, one would expect to get a considerable return in linguistic ability. Such is the subject of this chapter.

We will look at the major classes of English syntactic phenomena thematic relations, embedded clauses, questions and related phenomena, heavy phrase movement, ellipsis and other kinds of conjunction reduction, {{the verb group}}, pronominalization {{and discourse context}}. We will see how the different constructions are implemented, focusing primarily on how the exigencies of language production effect the way that the constructions may be analyzed, particularly the effects of variations at the message-level.

1. Special considerations in linguistic analyses for production

Every English construction, from the passive voice to reflexive pronouns to focus, is ultimately manifest as a particular pattern of open and closed class morphemes in a text. This much of the analyses is common to all kinds of linguistics whether it is done by generative grammarians, parser writers, or grade-school English teachers. However, beyond this simple description of sentential forms, the analysis of natural language constructions for use in production requires certain special considerations. Obviously there are differences in the kinds of representational devices available—the fact that this theory provides only one level on which purely linguistic relations can be arrayed ("extended surface structure") is already very significant: many grammatical facts which, in TG, would appear as facts about "deep structure" or "logical form I", appear here as facts about the structure of messages.

However, the more important differences do not involve the form of constructions so much as their function. The speaker has certain goals and employs the constructions it does entirely for the purpose of meeting those goals. It follow that every construction that is used in a text must be expressly motivated by the message.[1] This puts the constructions in a special light and requires us to determine as part of every analysis:

What kinds of message structures will motivate the use of this construction?

The answer to this question will tell us in what size "chunks" the message elements will arrive for embedding in the construction, and the chunk-size will tell us what kind of decisions we will be able to make on the basis of the chunks themselves without having to wait until their subelements become visible.

Still more important is the fact that the production process takes place over time. The linguistic component is not aware of all of the speaker's goals simultaniously, rather, goals become known as the message elements that represent them appear in a message's enumeration order. Thus another important part of every analysis will be to determine:

When will the need for the construction become known;
What will the state of the tree be at that time?

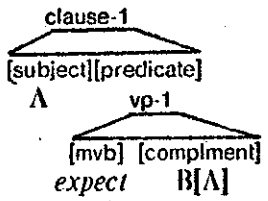
It would make a marked difference in the analysis of, e.g., an adverbial modifier whether its placement was decided upon at the same its matrix clause was created, or as an "afterthought" that happened while the clause was being traversed by the controller.

---

1. The one class of exceptions is constructions that are a grammatical reflex of some construction that is expressly motivated,. Subject-verb agreement is an example.

1.1 Ontological differences between domains

Obvious, but also very important, is the fact that the ontology used by different speakers/expert-programs ("domains") can be different. The analysis of the kinds of operations needed to bring about some construction will vary according to the conceptualizations involved. For example, two domains may both have the relation, expects(A B[A]), with dictionary entrys that cause the relation to be expressed as *A expects B* in both cases (as in *Macbeth expects to be king*), leading to identical configurations in the tree:



But one of the domains might conceptualize B[A] as a future event with a A as its explicit subject: (macbeth ako king), while the other conceptualizes it as an abstract predicate applicable to A: (lambda x. (x ako king)). Abstract predicates can be realized directly as verb phrases (e.g. *to be king*), but an event in the same context would require the application of a TRANSFORMATION to "prune away" its superfluous subject (see pg.<subject_pruning_transformation_n.y.w.>) and produce the same verb phrase. A very basic difference in the representational style of the domain is effecting the kinds of grammatical operations required to realize their messages.

1.2 The well-formedness constraint on messages

Not every message with the same semantic content can be realized using the same constructions. Because of the way that messages are interpreted, differences in pattern of intentional, communications-oriented goals in the message (or just in the pattern of semantic relations) can make a enormous difference in what realizations are possible. For example, of the two messages A and B below,[2]

2. Both of these messages are expressed as formulas in predicate logic and are intended to be processed as if by the dictionary of the logic domain (VII.B.2), i.e. depth-first decomposition of the formula according to the usual construction axioms of the predicate calculus.

[A] was_reported (is_likely (move_to_swiss_hospital (the-Shah, future-time)))

[B] say_about (the-Shah,
was_reported (is_likely (move_to_swiss_hospital (the-Shah, future-time))))

---

only message B is a possible source for a text about "*the Shah*", e.g.

"*The Shah is reported to be likely to move to a hospital in Switzerland.*"

This is true even though message A could be realized as any of the variants below (as could message B)

"It is reported (by government sources) that it is likely that the Shah will be moved to a hospital in Switzerland."

"Movement of the Shah to a hospital in Switzerland is likely[3], it was reported."

"That it is likely that the Shah will be moved to a hospital in Switzerland was reported (today)."

The kind of unbounded searching and transformation that would be required to understand message A as saying something about "*the Shah*" is not possible if the linguistic component is to be guarenteed to realize messages in linear time. Linear time depends on needing to make any decision only once, which in turn depends on the linguistic component knowing the relative importance and the relatedness of the speaker's goals, which information is, by design, to be encoded in the enumeration order. We can see this summarized in the well-formedness condition on messages, which is stated below.

The Well-formedness Constraint on Messages:

"Every message element must be realized before any of the message elements to which its realization refers."

$$\forall(m)\ m \in \{\text{messages}\}$$
$$\forall(e)\ e \in \{\text{enumeration}(m)\}$$
$$\underline{the}(r)\ r = \text{the\_realization}(e)$$
$$\forall(e_s)\ e_s \in \{\text{referred\_to\_by}(r)\} \wedge e_s \overset{\in}{\not\in} \{\text{enumeration}(m)\}$$
$$\text{preceeds}(e, e_s, \text{enumeration}(m))$$

For every message "m", for every element "e" in the enumeration order of m, the realization "r" of e must be such that e preceeds in the enumeration order of m each of those of its subelements, "e_s" that r refers to.[4]

---

3. Notice that it can be necessary to ignore highly subordinated goals (in this case realizing **future-time**) when earlier decisions create linguistic contexts whose grammatical constraints make those goals unrealizable. See section restrictions_on_realizations_n.y.w. for discussion.

4. If "realizations" are identified with choice-applications, then the function referred_to_by would return all of the message elements (if any) which are given as arguments in choice-applications. This is not, however, the actual way that the function is defined due to a technicality in the actual definition of "realization". [[?? do you want to know the real blood & gore ??]]

To see why message A cannot be realized with the Shah as [subject] of its major clause, we first consider why one might want to use that construction and when the linguistic component could become aware of those reasons. We will then see that because of characteristics of the component's behavior that the well-formedness condition captures, *it will not become aware of those reasons until it is too late to act on them.*

If the Shah is to be motivated as [subject], it will be either because of some property of the linguistic context in which message A appears, for example that the Shah is the current focus of the discourse, or because it is a goal of the message. In message B it is an explicit goal; in message A it would have to be implicit—the element the Shah might have an associated property (as as part of its dictionary ENTRY), e.g., that the Shah is always important and should be the [subject] of any sentence it appears in.

If the Shah were in focus, this would be signaled by a relation, either directly in the message or implied by one of its elements to the effect that:

(focus <elmt-in-focus> <msg-source-of-clause>)

which in this case will refer to the-Shah and to was_reported. If making the Shah the toplevel [subject] were a standing order, this would involve a CHOICE-FILTER in the ENTRY for the Shah, roughly:

(choice_must_map the-Shah into ([subject] current-major-clause))

The filter refers to a symbolic constituent structure position and thus indirectly to the message element whose realization incorporates that position—in this case again was_reported.

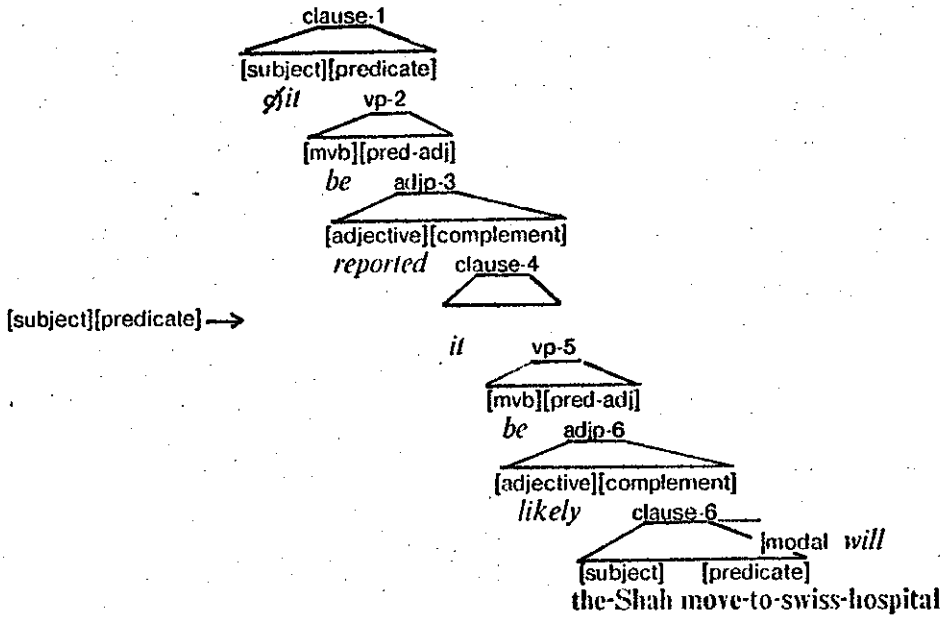
The enumeration order of the elements in message A (repeated below) follows from its relational structure, and is:

---

was_reported > is_likely > {move_to_swiss_hospital, future-time} > the-Shah
[A] was_reported (is_likely (move_to_swiss_hospital (the-Shah, future-time)))

Was_reported will be realized first. Since either of the motivations that would position the Shah as toplevel [subject] refer to was_reported, they must be noticed and acted on now if the well-formedness condition is to be satified. This however cannot happen given the structure of message A. The second case is ruled out immediately because the-Shah is three elements beyond was_reported in message A's enumeration order and must, therefore, be realized later. The first case is ruled out because by the principle of incremental realization (pg. <incremental_realization_n.y.w.>), TRANSFORMATIONS are able to refer only to immediate subelements, i.e. is_likely. The-Shah is invisible from was_reported and focus thus cannot apply.

The well-formedness condition is not a stipulation but a necessary consequence of the designed flow of control in the linguistic component. Consider what would happen if we applied the CHOICE-FILTER once the-Shah had been reached by the controller and there somehow was a CHOICE remaining that would do the mapping. The tree at that point would look like this (or some equally embedded equivalent).



Everything in the hashed region of the figure to the left of the arc has been traversed once by the controller and will not be traversed again. Consequently even if a change were somehow made in that region it would not change either the text or controller variable values since the controller will never "see" it.

Corollaries for message structure The linguisic component is intended for planned, fluent speech. Achieving this goal as an inevitable consequence of the design rather than by stipulation entails imposing constraints on the designer. Not every conceivable extended surface structure can be made to produce a desired construction just by writing the appropriate GRAMMAR-ROUTINES, and not every conceivable message can be made to produce a desired extended surface structure just by writing the appropriate ENTRYS.

The well-formedness condition reflects the fact that the message-level of representation is not the same as a semantic level or even the level(s) used by the expert program. It has its own motivation. As "message-builder", the speaker must be aware that the expressions it builds will be used directly to control the linguistic component's action. They will dictate the relative

importance of different goals through position in the enumeration order, and specify decisional dependencies through what subelements they make available to the ENTRYs in its dictionary. Message B (repeated below) has an enumeration order which gives the ENTRY for say_about access to enough of the message at once to be able to make a reasoned decision about what construction to employ (see section <blind_subject_predicate_choice> for details).

---

say_about > { the-Shah, (was_reported > is_likely
> {move_to_swiss_hospital, future-time} > the-Shah) }

[B] say_about (the-Shah,
was_reported (is_likely move_to_swiss_hospital (the-Shah, future-time)))

Even message B is less than optimal from the dictionary designer's point of view because of the "narrow" enumeration order that it has if the composition rules of the predicate calculus really are used. A structure that marked was_reported and is_likely as modifiers to the proposition and had all three elements visible simultaniously, would permit the Shah to be focused by a transformation and would provide maximum flexibility for the realization of the modifiers. [[?? I suppose you'd like to see how that's done ?? -- where in the text??]]

2. Thematic relations

Among the languages of the world, English is notable for its fixed word order. Language like Russian or Finnish with their extensive case-markings can present the arguments to a verb in almost any order, while English is for the most part restricted to the order: "subject, verb, (indirect object,) direct object, adjuncts". In Russian (and in other languages with a so-called "scrambling" rule), the alternative orderings are not synonymous, but reflect discourse-level relationships between the constituents, the speaker, the audience, and the rest of the discourse. Such *thematic relations* are also at work in English (though here they are harder to analyze) and are taken to be responsible for those order-adjusting constructions that English does have, i.e. passive, dative-shift, alternations orderings within the verb phrase, introductory adjuncts, topicalization,[5] and right or left dislocation.[6]

---

5. These last three constructions involve grammatical relations across unbounded amounts of text and are discussed in the section on WH-movement.
6. i.e. "*My daughter, she is such a wonderful girl*"—some noun phrase either sentence intial (left-dislocation) or final (right-) with a pronoun appearing where the noun phrase would have been.

This section will not present a theory of thematic relations. There are already many such theories in existance, developed in varying detail and from varying metatheoretical viewpoints, which could be adapted for production; see for example: [sgal_et_al][halliday_&_hassan] [martin_kay_functional_s_p][gruber_thematic_relations] [grosz_focus][sidner_thesis] [hobbs_coherence]. Instead, it will discuss how thematic rules (whatever they actually turn out to be) would be integrated into the production process. Two examples of possible thematic rules will be looked at: "focus" and the "given/new" distinction. The emphasis will be on at the point(s) where the rules are noticed and the kinds actions they control, rather than on the details of their analysis.

2.1 Focus

MUMBLE incorporates a very simple notion of discourse focus It has clear inadequacies, but is an interesting place to start; its intended use is in domains with minimal speakers that do little or no explicit planning. An ENTRY can specify that a given message element is "in focus" within a region of the tree. As a consequence, any sentence within that region will have the focused element as its [subject] (if it mentions it at all as an argument of the verb). Additionally, the pronominalization of a focussed message element will be encouraged (pg.166), and the element will be always taken as "given" (see below).

Properly written text is coherent; which is to say that it obeys certain conventions like the use of focus and given/new. As a designer, one would like to know how much of this coherency must be explicitly planned (i.e. present in the message) and how much will "fall out" from general principles. The implemented version of focus is designed as a "general principle" in that its effect is felt through the transformational system and is thus transparent to the dictionary designer. Its use in the "Macbeth domain" is typical. Suppose the speaker (specifically the *whole-frame_entry*) wants to produce a text that is focused on a particular character. What it does is to arrange that the source for the text (some sequence of FRL frames) is realized under one covering NODE of category *discourse*. That NODE is then marked with a "*focus*" HOOK whose value is the character to be focussed. The rest of the work is done entirely by the grammar.

Let us look at how this process was applied in the example from the introduction. There message involved a "sequence" of four properties (repeated below) dominated by the same NODE, and with a *focus* HOOK whose value is macbeth.

(macbeth (murder (duncan))) *;"murder-ma"*
(macbeth (become (king))) *;"ako-ma-2"*
(lady-macbeth (persuade (macbeth (act (murder-ma))))) *;"persuade-ma"*
(lady-macbeth (hq (ambitious))) *;"ambitious-lm"*

A "focus-transformation" applies as each property is realized, triggered by the *focus* HOOK and the fact that the realizations are simple clauses. In the first two instances, the entry's unmarked CHOICE is sanctioned because it already positions macbeth as [subject] (i.e. *Macbeth murdered Duncan in order __ to become king*). The unmarked CHOICE for persuade-ma, however, has lady-macbeth as its [subject] and must be transformed. Since the unmarked position of macbeth would have been direct object (i.e. [object1]), The grammar dictates that *passive* applies, producing "*He was persuaded to do it by Lady Macbeth*".

The last property, ambitious-lm, does not mention macbeth at all and might be a problem to "focus". However, there is a further option, namely to adjoin it to the phrase *Lady Macbeth* (the last noun phrase of the previous sentence) as a non-restrictive relative clause. Were it left instead as a separate sentence, the fact that it says something about lady-macbeth rather than her husband would cause a *de facto* shift in focus.[7]

The focus mechanism In order to notice potential violations of focus, those TRANSFORMATIONS that are capable of "moving" an element to [subject] are indexed according to the positions being "moved from" in the corresponding unmarked CHOICES—one CONDITION-SET per "bad" position. For example, *passive* and *extraposition-from-subject* can be triggered by these two "focus-detecting" CONDITION-SETs.[8]

```
((has-hook current-discourse 'focus) ;Is anything in focus?
 (argument-in-focus)  ;Is it one of ours?
 (equal 'object1  ;Is it the argument going to [object1]?
        (unmarked-constituent-for-arg (argument-in-focus)))
 (passive))

((has-hook current-discourse 'focus)
 (argument-in-focus)
 (equal '(predicate pred-adj adjective)
        (unmarked-constituent-for-arg (argument-in-focus)))
        (extraposition-from-subject))
```

Passive thus notices the focussed element being positioned in [object1], and extraposition-from-subject notices it going to ([predicate][pred-adj][adjective])[9]

---

7. Suspicions that "something is rotten in Denmark" are not ill-founded. As hinted at earlier, this framework for implementing focus as a "general principle" has definite limitations. First of all, its ability to transform texts to fit is predicated on being able to have both the unmarked position of the focus element and the [subject] position visible simultaniously. More importantly, because it is a linguistic principle, it is in no position to insure that the message-level sources for the text in fact are about the focus (which ambitious-lm certainly is not) It is really because of a deficiency of the particular message formalism and the planning (or lack of it) at that level that ambitious-lm appears in a sequence of properties about macbeth and is not already subordinated at the message-level (i.e. marked specifically as a description of lady-macbeth and left out of the sequence).

8. These TRANSFORMATIONS can, of course, have other triggers when they serve other functions.

9. Extraposition can be used to focus adjectives by rendering the [subject] semantically null: It's easy to please John.

Details: reasoning about CHOICEs symbolically

These CONDITION-SETs have first of all to be able to detect whether the current context includes a focus, and if so, which message element it is and whether it is one of the arguments of the CHOICE-APPLICATION being considered for transformation. This is done all in one operation by the PREDICATE *argument-in-focus*, which returns *nil* if any of those conditions are false and otherwise returns the parameter of the CHOICE which corresponds to the focussed element.

This choice of return value may at first seem obscure, but, in fact, it is forced on us. To perform this test we have to compare information given in two vocabularies: that of the message, and that of the CHOICE. The specification of focus is necessarily given in the message-level vocabulary because it identifies a specific message element. On the other hand, the specification of what elements will be mapped into what positions is given indirectly in terms of a vocabulary of parameter names local to the CHOICE. *Argument-in-focus* must determine how these two vocabularies correspond in a given instance. First it determines what element is in focus by looking at the *focus* HOOK of the *current-discourse-node*. It then looks at the CHOICE-APPLICATION being considered for transformation. The application's arguments are LOCAL-VARIABLES of the ENTRY being interpreted (part of yet a third vocabulary). At this point in the realization process, the message elements which correspond to these variables are known, and *argument-in-focus* can determine if the focussed element is one of them. Assuming that is so, it now symbolically simulates part of the process of applying CHOICEs to their arguments, and by this determines which of the CHOICE's parameters corresponds to the ENTRY variable bound to the focussed message element. The predicate unmarked-constituent-for-arg takes that parameter, goes to the CHOICE's map property, and looks up which constituent position corresponds to that parameter.

Other focussing constructions

Certain constructions syntactically mark a constituent as focussed. The text below (from [candy_thesis] page 90) uses *there-insertion* in sentence one to introduce *strawberries* as the focus, and then uses *cleft* in sentence four to shift the focus to *Mark*. Notice how the interveening sentences use pronouns for the focussed items and make them the [subject] wherever pragmatically possible.

1. *Last week there were some nice strawberries in the refigerator.*
2. *They came from our food co-op and were unusually fresh.*
3. *I went to use them for dinner, but someone had eaten them all.*
4. *Later I discovered it was Mark who had eaten them.*
5. *He has a hollow leg, and it's impossible to keep food around when his stomach needs filling*

The constraints on MUMBLE's design are such that it is not possible to treat focus-introducting constructions in the same way as "focus-maintaining" ones—there is not way to

create a hook with the meaning: "choose some construction that would introduce *strawberries* given this matrix".

2.2 Given/new

The given/new distinction is a standard part of the functionalist's repetoire (for example see [given_new_references]). It is usually analyzed on a sentence by sentence basis, and divides the text of the each sentence into an initial segment which is *given*—relating information already known to the audience, and a final segment which is *new*—information that is being introduced for the first time or highlighted. Again, only a very simple notion of given/new has been implemented in MUMBLE as an automatic part of the grammar. MUMBLE will notice when a message element is "given", and choose between pairs of constructions on that basis.

The analysis of given is entirely analogous to the one for focus: We would like the text to reflect some principle without having to redundantly state it in every dictionary ENTRY. To do this we must (1) create at test to tell when the principle applies (for focus this was the presence of a *focus* HOOK), and (2) determine where the principle is to have its effect and thus where to position the test. (for focus this was during the evaluation of TRANSFORMATIONS controlling constituent order within the clause).

The present test is crude: an element appearing at some position in a text is "given" at that position if it has been mentioned before and is not "stale"—the basic conditions for pronominalization. As with focus, given has its effect during the evaluation of transformational alternatives to an ENTRY's choice; this at the moment influences three specific transformational-families (immediately below) and one schematic one (next subsection).

particle movement: "*...picked up the ball* vs. *picked it up*"
possessive/genitive alternation: "*The subconcepts of Pp* vs. *Pp's subconcepts*"
dative shift: "*...give a problem to Dan* vs. *...give him a problem*"

In keeping with the marked/unmarked style of analysis that I have adopted throughout the grammar, one member of each family is taken to be the unmarked "default": in this case the ones on the left. [[why??]] The transformation to the marked form is triggered when a particular one of the message elements involved is noticed (by the grammar) to be given (these are the direct object, the object of "*of*", and the indirect object respectively).[10] The same style condition-sets are used as for focus: the critical message element, identified by its position in the unmarked construction, is tested for being given, and if that is the case, the transformation is applied. Below is the TRANSFORMATIONAL-FAMILY for the possessive/genitive alternation.

---

10. Some readers will recognize these three transformations from the linguistic literature where they are typically analyzed as "obligatory" when the shift-able constituent is a pronoun. They are glossed here here as under the control of a general thematic principle because (1) pronouns are just an extreme form of "given" information, and (2) when we look ahead to the time when more thoroughly planned input is available from speakers, "given" may be a label attached when a message is assembled—making the assessment even easier than it is now.

```
(define-transformational-family  possessive-genitive
    ((given  (argument-that-will-be-in 'of-obj))
     (of-obj= =>determiner)) )
```

Using "given" inside entrys

An important aspect of coherent discourse is that it does not repeat the obvious. Insuring this is, for the most part, a matter of exercising pragmatic judgement while assembling the message: the speaker examines its model of the audience and leaves out those facts that it believes they already know. However, it has a linguistic aspect as well, since it is usually safe to assume that the audience knows everything that has been said recently.

To a first approximation, an entry can safely omit any parts of a description that are "given" by warrant of having been recently mentioned. Alternatively one can use phrases like "*another*" or "*such*", and, in the limiting case, use a pronoun—indeed, this whole process of reacting to "given" information is another way to look at subsequent reference. The difference is that the subsequent reference routine is general purpose,i.e. it depends entirely on relations that can be formalized uniformly within the linguistic component; on the other hand, individual entrys can be written so as to take advantage of their special semantic knowledge by testing for and reacting to "given-ness" directly.

We can see some examples of the possessive/genitive alternation and individual "omission" decisions by ENTRYs in the first two paragraphs of the KLONE-net-as-object example (repeated below). The relevant phrases are picked out by superscripts.

> *Phrase is the top of*[1] *the net. Its*[2] *interp role must be a concept, and its*[2] *modifier role and its*[2] *head role must be phrases. Its*[2] *subconcepts are pp, np, adjunct, indobjclause, and word.*
>
> ___[3] *Pp has the roles: pobj, prep, interp, and ppobj. Pobj*[4] *must be a np, prep*[4] *a prep, interp*[4] *a relation, and ppobj*[4] *a pp. Pp's subconcepts*[2] *are ofpersonpp, insubjectpp, locationpp, and aboutsubjectpp..*

Phrase (1) appears in the genitive because this is the beginning of the discourse and *net* has never been mentioned before—it is "new". All of the (2) phrases, however, appear in the possessive because their "possessor" (the concept *phrase* in the first paragraph and *pp* in the second) is "given" by warrant of being focus of the paragraph and thus emmenently pronominalizable. For comparison, see how disjointed the text is when the genitive is used instead:

Phrase is the top of the net. The interp role of phrase must be a concept, and the modifier role of phrase and the head role of phrase must be phrases...

The phrases marked (3) and (4) are interesting because of what they leave out. (3) marks where the sentence *Pp is a subconcept of phrase* would have been except that the sentence just before it has already listed the "subconcepts of phrase", making the content of (3) "given" and therefore redundant. Similarly, the descriptions of roles marked as (4) are proper names rather than noun phrases (e.g. *the pobj role of pp*) because the fact that they are "roles of pp" was given by the summary sentence that begins the second paragraph.

Superscript (3) marks the omission of the sentence "*Pp is a subconcept of phrase.*". This was an explicit option of the *describe_sub-concepts* DECISION of *concept-defining_entry* (pg.<concept_defining_entry>). Similarly the (4)'s mark the omission the fact that "pobj", "prep", etc. are "roles" and furthermore roles of the concept pp—that fact having been already given by the summary sentence that started the paragraph. (The regular entry used for roles in shown on page <role_entry_n.y.w.>.) An alternative way to arrive at the same decision would be via a subsequent reference heuristic to the effect that once an object has been referred to via a proper name, that name should continue to be used until the reference is stale (pg.<name_name_heuristic_n.y.w.>).

Given/new in the answers to questions The primary difficulty with moving more of the responsibility for given or other thematic relations into the grammar and out of "hand-crafted" ENTRYs is the lack of effective formal criteria to decide when the relations apply. The linguistic component cannot on its own apply any semantic or pragmatic criteria because it knows nothing about them—it does not "understand" what it is saying in any significant sense.

Another, quasi-syntactic way to know that some fact is "given" is for it to be part of a question that the speaker is answering. None of the micro-speakers that I have developed answered questions (none of them ever had dialogues), but it is easy enough to construct an example for purposes of illustration.

Anticipating a later example of question formation, we can imagine that the KLONE-based program of section III.4.3.i has been asked the question:

"*What states do jump arcs go to from S/NP?*"

Note the order of the prepositional phrases inside the verb phrase: first "*to*" then "*from*"; this order should be reversed in a full-sentence answer:

"*Jump arcs go from S/NP to S/DCL.*[11]"

This order concentrates the given information at the beginning of the sentence and positions the new information—the answer—at the end, the unmarked position for sentence stress.

In order for the order of arguments within the verb phrase to be subject to transformational control, the alternative orderings must be part of a transformational-family, and given/new must be cast as a reason for deviating from the unmarked order. For example:

```
(define-transformational-family  go-from/to
    ;the unmarked order is taken to be "...go from A to B"
    ((and (given from-arg)
          (not (given to-arg)))
     (go-to/from)) )
```

(Note that by serendiputous chance, the unmarked form already has the appropriate order in this example and the transformation does not apply.) By using a transformational-family—a procedural device—to represent these facts, rather than some kind of declarative constraint, I have effectively decided that no decision-maker will need to know about this constraint, but only to behave in accordance with it. That is, when the given/new constraint is represented by a transformational-family, the decision-makers can not know about it even should the designer (changing his mind) want them to, because the information is not in a manipulable form: it has been compiled into the decision-trees of the entrys that select the affected constructions. Performing the compilation makes the process faster because there is less interpretation involved, but more important is the fact that it makes the decision unconscious. Perhaps it should not be: perhaps in certain modes of production one is able to reason in terms of given/new alternatives. If so, the design will have to be changed. Regardless of what turns out to be the case after further analysis, couching the difference directly in terms of incompatable data-types of the theory is an effective design heuristic that fosters clear distinctions in hypotheses.

11. A better way to phrase this answer would probably be to say: *"Jump arcs from S/NP go to S/DCL"*. This bunches the "given" information more closely and avoids the impression somehow imparted by that answer that it is giving us the information for the first time. However, shifting information from a predication to a reference is too much of a semantic shift for a linguistic transformation to perform and would have to be done earlier at the message-level.

3. Embedded Clauses

The grammar of embedded clauses is characterized by idiosyncratic details and nearly synonomous alternatives. Consequently we would like it to be as transparent to the dictionary designer as possible—something that they would get "for free" by using this linguistic component to construct their texts. I will begin by describing the facts that their analysis must cover, then show how the facts are represented in MUMBLE's grammar generally and for specific cases, including what the designer has to specify when adding new vocabulary. Finally, I will look at a general problem of message interpretation that these constructions provide a clear example of.

3.1 The facts

A clause may appear either by itself (i.e. as an "independent sentence" or "main clause") or as a constituent of another, "higher" clause, in which case we will refer to it as an *embedded clause* (also "dependent sentence" or "subordinate clause"). Embedded clauses can assume essentially any clausal function except that of verb. Thus we can have sentential subjects:

"*For Macbeth to murder Duncan was a horrible deed.*"

direct objects:

"No one knew that Macbeth murdered Duncan."
"MacDuff feared the man who[12] *murdered Duncan."*

"complements" to verbs, prepositions, adjectives, and nouns:

"Lady Macbeth persuaded Macbeth that he should murder Duncan."
"They swore vengence upon whoever had murdered Duncan."

and clause modifiers:

"By murdering Duncan, Macbeth became king."
"Macbeth became king because he murdered Duncan."

Looking at these examples, all alternate realizations of the FRL expression: (macbeth (murder (duncan))), we see that the content of a clause when embedded is essentially the same as when it acts as a main clause: the order of its constituents is unchanged, as are the words that can be used. This suggests that the proper analysis of these constructions should involve TRANSFORMATIONS. That is, all of the variants of a clause (including, as we will see later, questions and thematic variations) should share the "early" portion of a clause-building ENTRY's decision-procedure, i.e. the choice of verb, number of objects, and the mapping from subelements to constituent positions, and should differ only in what "later" TRANSFORMATIONS apply.

12. Embedded clauses based on relative clauses (such as this one and "*whoever...*" below it) involve grammatical relations over unbounded distances and are discussed in the section on wh-movement.

Another observation is that there are many different syntactic and mophological alternatives for the form of an embedded clause. These are correlated to a certain extent with the different embedding positions, but there are usually still several alternatives for each position, alternatives which, to the casual observer, appear to be synonomous:

"That Macbeth murdered Duncan was central to the story."
"For Macbeth to murder Duncan was central to the story."
"Macbeth murdering Duncan was central to the story."
"Macbeth's murder of Duncan was central to the story."

The "synonomy" is not complete however. The alternatives differ in their possibilities for expressing tense, aspect, and modality; they assign different, if subtle, degrees of "agent-hood" to *Macbeth*; and their metrical structures are very different. Consequently the analysis must (1) be able to discriminate among the different embedding positions and do so in a way that allows them to be annotated according to what alternatives they permit, and (2) provide a facility to the designer for making specific selections when there is a reason.

The analysis From the point of view of the dictionary designer, ENTRYs choose "clauses", not "major clauses" or "embedded clauses".[13] That distinction is a grammatical one, embodied in the TRANSFORMATIONAL-FAMILY "embedded-clause-transformations". This family is assigned to every CHOICE whose phrase mentions the SLOT-NAMEs subject and predicate. It is where the different embedding contexts are detected and where the selection of a suitable embedding transformation is staged.

If a clause is to be embedded, this will be indicated by the current-slot: subject, object, adjunct, and complement are the labels currently used in MUMBLE. PREDICATEs in the CONDITION-SETs of embedded-clause-transformations look for these specific constituent-structure-labels and select corresponding grammatical-decisions to make the selection of embedding transformation.

13. Note that after postprocessing, the set of possible matrix choices of an entry is effectively the transitive closure of the choices provided by the dictionary designer and the transformations associated with them in the grammar. Therefore in the run-time environment this statement is no longer strictly true.

```
(define-transformational-family  embedded-clause-transformations

    ((is-slot 'subject)
     (sentential-subject-possibilities))

    ((is-slot 'object)
     (sentential-object-possibilities))

    ((is-slot 'adjunct)
     (sentential-adjunct-possibilities))

    ((is-slot 'complement)
     (complement-decision-for  current-mvb)) )
```

Note that while the first three contexts needed no additional information to select the correct set of alternative constructions; [complement] contexts are further specified according to properties of the *current-mvb* (see below).

At this writing, none of the existing micro-speakers have been able to motivate conceptual distinctions among the alternative embedding transformations. Consequently, all that the selected grammar-decisions do is to use a general-purpose {entry-level} subroutine for managing sets of synonomous constructions (see section synomym_set_n.y.w.). The one exception to this is a check to see if the speaker explicitly wanted to express tense or modality in the clause, in which case the "report clause" construction is selected when it is included in the synonym-set. The dictionary designer may augment or override the "default" GRAMMATICAL-DECISIONs by replacing them with new ones incorporating whatever usage-heuristics the designer wants. Below is the decision for sentential subjects.

```
(define-grammatical-decision  sentential-subject-default
    default  (synonym-set '(that_report
                            poss_ing
                            for_to
                            obj_inf
                            participle))
    ((has-attachment  instance-being-realized
                      '(tense modal))
     (that_report)))
```

Commonalities in the transformation and refinement of embedded clauses are captured by having the individual TRANSFORMATIONs refer to common editing subroutines and employ the same constituent structure labels in their output.

I will now go through each of the embedding contexts and subordinating alternatives as presently analyzed in MUMBLE. The specific mechanisms that bring about the different morphological patterns will be explained and the assumptions about message-level chunkings will

be discussed.

3.2 Sentential subjects

Any clause-specifying CHOICE selected for an ELMT-INSTANCE in a [subject] SLOT must be transformed by one of *that_report*, *obj_inf*, *for_to*, *gerund*, or *poss_ing*.

that_report

There is no grammatical difference between properties of a major clause and of an embedded clause complementized by "*that*" beyond the complementizer itself. Consequently the TRANSFORMATION that_report is quite trivial: it merely adds the NODE-FEATURE optional-that[14] to the features of clause-node being specified by the choice undergoing transformation:

```
(define-transformation  that_report
    phrase ((add-feature-to-schema-instance 'optional-that
                                            'basic-clause))  )
```

obj_inf

The only difference in the grammatical properties of a major clause and one embedded using *for_to* involve the morphological properties of its [subject] np (which appears in the oblique case rather than the nominative) and of the "tense-carrying" verbal element (which appears in the infinitive obligatorily preceeded by the function word "*to*"; neither tense nor modality can be expressed, although aspect can). In particular, as in a major clause, subsequent references to the "subject" message element must be realized as reflexive pronouns (pg. <reflexive_pronouns_n.y.w.>), and the verb may take a [complement]. These facts (among others) imply that if the grammar is not to be unduely complicated, the CONTROLLER-VARIABLES associated with major clauses must be present with their normal meanings.

The analysis is again to supplement the usual constituent structure labels with specializing features.[15] To the [subject] we add the feature oblique, and to the [predicate] we add the feature infinitive. Both SLOT-FEATURES are signals to the morphology routine: they create marked contexts that override the unmarked, default morphological reflexes to always select pronouns in

14. Actually, when that_report is applied in a [subject], the initial "*that*" is not optional: it must be present if the audience is not to "take a garden path" and missinterpret the sentential subject as the main clause of the sentence. However, there is something to be said for using the identical transformation in both [subject] and [complement] (where the *that* is optional). In any event, a conditional test against the current-slot would be required somewhere regardless of the analysis, and I have elected to place it in [optional-that after-realization] which is the routine tht will print the "*that*" if it is included.

15. Note, these features will appear in the features property of the local SLOTs; they do not change the permanent list of features of the SLOT-NAMEs.

the nominal case and to express the tense of the clause on the first element of the verb group.

```
(define-transformation  obj_inf
    phrase ((add-feature-to-slot 'oblique
                                 '(subject basic-clause))
            (add-feature-to-slot 'infinitive
                                 '(predicate basic-clause)) ))
```

The slot-feature infinitive also filters out any CHOICEs that would add HOOKs for tense or modality to the constituent in the [predicate]; see section <filtering_out_parts_of_the_message> below.

for_to

The for_to construction is the same as the obj_inf except that now the [subject] is preceeded by the complementizer "*for*". It can thus be implemented simply by taking the specification for obj_inf and adding an instruction to add the feature say_for to the [subject].

```
(define-label  say_for
    enter-slot
        (lambda  (contents)
            (mprint '(for)))
```

gerund & poss_ing

A gerund can be thought of as a verb phrase with the surface syntax of a noun phrase. It can employ certain of the same determiners: "*your singing in the shower is waking the neighbors*", and can be qualified with a relative clause: "*The sacking of Rome that accompanied the migration of the Visgoths in the 5th century A.D. was particularly thorough.*".

```
(define-transformation  poss_ing
    phrase  ((replace_A_with_B  basic-clause regular-np)
             (add-feature-to-slot 'subject
                                  '(determiner))
             (add-feature-to-node 'clause
                                  t) ;add the feature to the top node (the NP)
    map (((subject) . (determiner))
         ((predicate) . (head))
         ((adjuncts) . (qualifiers))) )
```

If the clause-specification of the ENTRY's CHOICE includes a [subject], it is mapped into the [determiner] slot where the morphology procedure will cause it to be marked as possessive. If there is no [subject], a default-decision associated with the [determiner] slot (pg.<default_decision_for_determiners>) will be made and select a determiner on its own. A [subject] may be "missing" either because the ENTRY decided that it was not relevant for this instance, or (what amounts to the same thing) because the source of the gerund was an "abstract predicate"—an actor-less action whose conceptualization in the expert program never included a "subject"; see III.2.3.

Because the addition of the labels clause and subject effectively create a clausal context (at least as regards controller-variables) the reflexivization routine will apply in its normal fashion when a subject is present:

"*Mayor White denying himself a raise twice in a row is very unlikely.*"

However, subject-less predicates can pose a problem.

"*In an election year, giving himself a raise is a dangerous move for a politician.*"

The grammar insists on certain minimal information in order to determine the morphology of the reflexive pronoun, specifically the person, number, and gender of presumptive subject. Domains whose representations provide generic default information (such as KLONE) can provide this sort of information in a natural way, others will require possibly awkward *ad hoc* extentions to their dictionaries or may be forced to avoid the use of gerunds with reflexives entirely.

nominalization

The distance between a gerund and a true nominalization is relatively small: essentially a transformation from the unmarked direct object to the genitive relation and a dramatic extention of the morphological options available to create nominal forms of the verb beyond just "*-ing*".

gerund: "*Macbeth's murdering Duncan was the key action of the second act.*"
nominal: "*Macbeth's murder of Duncan was the key action of the second act.*"

To the extent that the nominal form of the verb is predictable (or is consistent—it can be listed as an explicit property analogous to irregular plurals or past participles), nominalization can be treated as a transformation transparent to individual entrys. Except for the different source of the actual nominalized verb, its form is essentially the same as the transformation for poss_ing.

However, I have not chosen to include nominalization with the other clause-embedding transformations because its difference in "emphasis" or "perspective" seems be too large to truely be transparent to an entry's semantically-based decision. In the one domain where it is presently available as an option, the Macbeth domain, nominalization is selected only in response to a specific attachment on the source message element that indicates that it is to be viewed as a unitary "event" rather than an action by an actor.

3.3 Sentential objects

Semantically, most of the sentential arguments to verbs are propositions, events, or actions rather than objects. They are analyzed here as appearing in a slots marked with the slot-feature complement (discussed below) while proper objects appear in slots marked object. Object will apply either a noun phrase or a so-called "headless relative":

"Put a red block into the box."
"Put into the box whatever blocks are at the front of the table."

This, of course, is not a transformational distinction but rather an alternation that is available to an entry. The headless relative is appropriate whenever an indirect description of the object is desired or is all that is possible. In heavily planned messages, these two alternatives would have different message-level sources. (The headless relative construction involves "wh-movement" and is discussed in section III.3.6.)

Verbs such as "*like*" or "*know*" that take both objects and propositions are analyzed as having both slot-features as features of its [object1] slot.[16]

```
(define-choice like_clause (s o)
    phrase (basic-clause ()
                predicate (vp-object1 ()
                                mvb like
                                object1 features (complement object) ))
    map ((s . (subject))
         (o . (predicate object1))) )
```

{thus} depending on whether the message element contained in this [object1] is realized as a noun phrase or a clause, one of the two features will have no work to do.

3.4 Complements

All of the TRANSFORMATIONS that apply to sentential subjects are also possible for complements. However, individual verbs will typically take only a few alternatives, their particular set being indicated by the property complement-type that every verb used with a [complement]-including constituent-schema must have. The value of this property is always a grammatical-decision; a declarative enumeration of a verb's alternatives (should one be wanted) is always available from the inversion of the decision that is computed by the postprocessor at load-time (see pg.182) [Complement]'s have one additional transformation that they do not (and could not) share with [subject]'s, namely "equi".

---

16. Alternatively one could have two "conceptual" sources for such verbs.

Equivalent noun phrase deletion

Equi is one of the "classic" transformations of transformational-generative grammars. There it deleted the subjects of embedded clauses under lexical identity with the subject of the main clause (see, for example, [burt][Permutter_&_Soames]). It performs essentially the same function in MUMBLE—reducing embedded subjects—but with one critical difference: here it performs its comparison at the message-level rather than at a linguistic level. [[elaborate on the differences?? -- mumble's version applies much more widely, subsuming subject-/object- raising as well]]

The message element that would be mapped into the [subject] of the clause being embedded is compared with the *current-subject* (or with the *current-direct-object*, depending on the verb). If the two elements are the same, then the new, would-be [subject] must be supressed, i.e. it may not appear in the output text. This can be done either by arranging for it to be realized as a trace, or by pruning the clause transformationally so that only the contents of its [predicate] are actually returned as the realization. These two techniques differ (1) in when they can be effected—the trace could be laid-down as at late a point as the realization of the subject, while the transformation must take place during the realization of the source for the embedded "clause", and (2) in the "derived constituent structure" that they leave—in the one case there is a clause node, in the other there is not. These variations propagate through to the statement of other grammatical rules, particularly reflexivization. I have opted for the pruning transformation because it has the action closest to the point of decision.

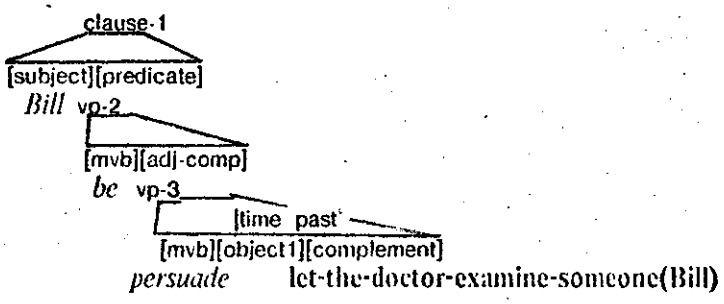
Equi is the only transformation to apply to [complement]s and not to [subject]s. It is also one of the few that is obligatory: if its conditions are met, it preempts all alternatives except that_report. This fact must be reflected in the order of the condition-sets within the complement-type decision, as is whether the comparison is made with the matrix [subject] or matrix [object].

An interesting case arises with verbs whose control relation (i.e. whether the comparison is made with [subject] or [object]) can change depending on their voice. The [complement] "*persuade*", for example, is controlled by "*persuade*"'s [object] in the active mode ("*we persuaded Bill ___ to let the doctor examine him*"), but by the [subject] in the passive ("*Bill was persuaded ___ to let the doctor examine him*"). There are two ways of dealing with this.[17] Either we remember what logical case "*Bill*" had (i.e. which number argument it was to its containing relation at the message level), which would be the first time such a thing had been needed, or we have the grammatical-decision actively look for whether or not the passive had been applied as it is running.

---

17. Actually the simplest way is when the speaker conceptualizes "persuade" as three place predicate involving a final predicate, rather than a final event, e.g. **lambda x. let-the-doctor-examine-somebody(x)** rather than **let-the-doctor-examine-somebody(Bill, time-t)**. In such a case the "equi" has already been done in the construction of the message.

Yet another possibility for encoding this dependency would involve redefining where the grammatical-decision was to be found. This possibility is particularly interesting because it illustrates the potential of the constituent structure as a succinct representation for implicit rules. Consider that adjective complements, just like persuade in the passive, always involve control by the [subject], e.g. "*Bill is easy to please ___.*" If we said that the controlling verb in the passive case was not "*persuade*" but "*be*", then we would have the effect we need without requiring a special, *ad-hocf0 check within the decision since complements of "be" are always controlled by the [subject]. (N.b. the constituent structure of the passive is taken here to be the same as the structure of a predicate adjective; see the snapshot below.)*



How then does the predicate used by equi know what verb to look to for its instructions?—it uses the controller-variable current-verb, which is maintained by [mvb leave-slot]. If that grammar-routine is made sensitive to its context such that, say, it would recursively reset the variable only if the clause was tensed, then it would "automatically" point to the verb with the correct control relation.

3.5 Sentential adjuncts

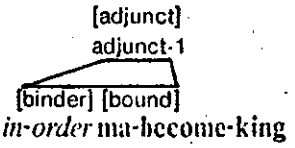
Adjuncts appear either with or without initial subordinating conjunctions or prepositions. If a subordinator is present, it will dictate the set of embedding transformations available, using the device of a complement-type hook completely analogous to that used for verbs and their [complement]s. The principle difference is that equi is no longer obligatory.

"Before he got married, John was a gad-about."

"With gold reaching the prices it has, dentists are looking for new ways to cap teeth."

"In order for us to continue to make a reasonable return, we must regretably raise our prices by 15% during the next calandar year."

These adjuncts are analyzed either as two-slot "adjunct" nodes where the "binding" conjunction appears explicitly, or as the contents of special slots where the conjunction is effectively part of the syntactic context. The choice depends primarily on whether the conjunction has an explicit source at the message-level.



[in-order_adjunct]
ma-become-king

When there is no subordinating conjunction the embedded [subject] is invariably[18] the same as the matrix [subject] and equi applies obligatorily. The remaining verb phrase takes a participial form, rather than the infinitive form it takes in complements. (The implementation of this grammatical detail is discussed in the section on the verb group, III.2.4.)

In some domains (e.g. Macbeth), the chunking of the message is such that adjuncts and "main-clauses" are introduced into the tree as sibling constituents (example on pg.<backwards_anaphora_snapshot>). In these cases, introductory adjuncts are completely realized before the controller ever reaches the main [subject]. This raises the question of how equi can be tested for within the introductory adjunct without a [subject] to compare with. If the check is to always be made, then the answer is "only with considerable planning". However, equi is optional in these contexts, one could easily believe that people only use the construction when in fact they have planned what the main [subject] will be; this analysis would remove reduced introductory adjuncts from the list of option stylistic constructions to be used when possible and make them keyed to specific rhetorical goals in a message.

3.6 Restrictions on realizations

The linguistic component cannot be aware of all of the speaker's goals simultaniously—the decision made for one message element is not the result of a gestalt appreciation of all of the other elements in the message but only of those that are realized before it. The properties of natural languages as representations make it inevitable that the number of available realizations for more and more deeply embedded message elements will be reduced and constrained, so much so that some of the speaker's later goals may have to be abandoned as unrealizable. Consequently, more important goals (message elements) must preceed less important ones in the enumeration order.

An example of this is the earlier case where the time-future specification of message A (pg.86) was omitted in an embedded clause. time-future is intended to be realized by the model auxiliay "*will*", but the clause it would have been part of was pruned away because it appeared in

18. As with similar "proscriptions", this grammatical fact can be implemented in two ways. Most directly, a choice-filter can be associated with adjunct slots that would inhibit it contents from being realized as an unbound clause. This has the drawback that it may be too much to expect the effected entrys to have both bound and unbound alternatives available to them—such a filter might often as not inhibit any realization at all. Alternatively, one can attempt to insure that only bound clauses (or their message-level precursors) are ever put in an [adjunct] {slot}. As [adjunct]s are clause-modifiers semantically, this means insuring [[by...]] that only {propositions} with a lexically realizable relation to the "main" clause are even used.

the [complement] of *is_likely*, thus forcing the main verb to be in the infinitive which is grammatically incompatable with a modal. If potentially-abandoned elements are available early enough, then a reasoned decision can be made. (In this case, is_likely must be visible when the decision is made about the choice of complement structure for the realization of move-to-Switzerland, which it is.) If this is not possible or if an explicit choice is made to abandon the element (as above), then CHOICE-FILTERs in the grammar will apply and eliminate all of the realizations available.

[[[provide an explicit example/snapshot of this filtering ??]]]

People will often use these grammatical restrictions to advantage: I can recall once while planning a talk coming up with the sentence "*the speaker knows what he wants to say*", only to get a negative reaction from some internal editor to the effect of "why should the speaker be male?". I then tried again and got "*...what she wants to say*". That prompted an equally negative reaction, and lead me finally to "*...what to say*" which settled the problem by keeping it from coming up. In a program it would be more natural to have all three possibilities equally available from the start as a set of choices and to then deliberate between them as a group. The fact that most people apparantly do not have them equally available and instead must enumerate them in a mode of generate and test may signify something about the human language facility.

A further fact about subordinated clauses is that so-called "root transformations" such as topicalization, tag questions, clefting, or preposed directional adverbs [emonds_root_transf] cannot apply to them. For example, one cannot say:

*"*Peter expects on the left there to be an antique lamp."*
*"*He went in the springtime where you like to go."*

but instead have to say:

"Peter expects there to be an antique lamp on the left."
"He went where you like to go in the springtime."

This grammatical rule can be enforced easily enough through the use of CHOICE-FILTERS: we label every clause-embedding context (except [complement] which can take report clauses not subject to this constraint) with a feature—*no_root-transformations*—and associate with it choice-filters attuned by name to the individual root transformations

However, it is not obvious that in fact one wants root tranformations to be filtered out unthinkingly. Consider that with a speaker that throughly plans its messages root transformations (which are always the marked case) would not be considered without some specific message-based motivation. In such cases, rather than apply a filter automatically a better idea would be for the dictionary designer to explicitly anticipate the places of possible grammar—goal conflict and to develop decision procedures to handle them.

4. WH-movement

For the constructions of the previous sections, the linguistic component was required to implement or test for grammatical relations that involved constituents within the same clause, or, at most, within two hierarchically adjacent clauses. This section looks at an important class of constructions, those involving so-called *WH-movement*,[19] whose implementation requires sensitivity to grammatical relations spanning nearly arbitrary distances. Such constructions could [be expected to] pose a problem [in this framework] because, on their face, they require a kind of processing that is antithetical to the incremental processing philosophy of this linguistic component.

The analysis that I will propose for wh-movement makes particularly strong demands on the speaker to provide a suitably indexed mesage-level source structure. I will argue (1) that, however acquired, the index information is an absolute requirement if the construction is to be employed as broadly as it is by people, and (2) that even if we ignored the limitations of the linguistic component's control structure, the needed information can be more easily obtained by the speaker than by the linguistic component. This kind of argument has strong implications for what would constitute an adequate design for a speaker program.

4.1 The basic facts

Wh-movement is a part of many different English constructions. All of them involve clauses; all of them have an initial "wh-phrase" in a pre-subject position that has no counterpart in the constituent positions of an unmarked (declarative) clause; and all of them have a "gap" (or an obligatory resumptive pronoun) at some point within the clause which, in the unmarked clause, would be filled by a phrase that corresponds to the wh-phrase.

question: "*Who was Macbeth persuaded to murder ___?*"
relative clause: "*Anyone <u>that Guiseppi shaves</u> ___ does not shave himself.*"
headless relatives: "*<u>Whoever ___ doesn't shave himself</u> is shaved by Guiseppi.*"
generic relatives: "*<u>Anyone who doesn't shave himself</u> is shaved by Guiseppi.*"
topicalization: "*A "B" I would never have expected to get ___ in that course.*"

---

19. This term originated in the transformational literature of the early 1970's [wh_as_movement_rel], at which point it had a literal meaning in terms of the transformational cycle. It is a useful term because it ties together otherwise diverse constructions according to a common grammatical property. In the analysis of the time, questions &et al. originated as sentences that had a "WH" word or symbol (i.e. &who, &what, &where, &when, &which, or &how) embedded in the position that the equivalent np would have had in a declarative version of the sentence. With sucessive applications of the bottom-up transformational cycle, this WH item then moved up through the sentence (either in one leap or in per-clause stages according to the particular analysis) until it reached its ultimate destination. More recent analyses (e.g. [wh_as_control]) have dropped any notion of explicit motion in favor of a notion of a grammatical "control" relation that holds between the sentence-inital WH phrase and a variable at the embedded position.

left dislocation: "*That stranger, your brother should never have loaned him any money.*"
Cleft: "*What I got ___ in that course was a "B".*"
pseudo-cleft: "*It was a "B" that I got ___ in that course.*"
"tough" movement: "*Micro-economics is tough for undergraduates to take ___ their freshman year.*"

The wh-phrase and the gap have grammatical properties and restrictions on their occurance that are of critical importance to the details of the implementation and of how they are planned by the speaker. These will be considered after the basic parts of the analysis have been presented.

Common semantic basis As diverse as these constructions are, they share a common semantic basis. They involve the abstraction of some item from a proposition in order to create a description. This abstraction process can be used as a way to avoid having to mention the abstracted item;

"*Put this block where you put the last one.*"

or as the basis of a question:

"*Where did you put the last block?*"

or as a predicate:

"*The box is where I put the last block.*"

or as a description:

"*...the place where I put the last block.*"

In every case essentially the same semantic operation has been performed, with the construction taking different syntactic forms depending on the function to which it is put.

The point is that, because of this common semantics, we should expect the message-level source for wh-constructions to always be a relation over a *pair* of message elements: the wh-item and the proposition it was abstracted from. The form that the two elements can or should take is another question entirely and, as we will see later, depends on the kind of information the grammar requires and the sophistication of the speaker's representation. For present purposes, I will assume a straight-forward, assertion-based representation like the one used in Winograd's SHRDLU program (based on MICROPLANNER [microplanner_manual]) or in my version of the KLONE-nets-as-objects domain. The message source for the last set of sentences would then be some variation on this:

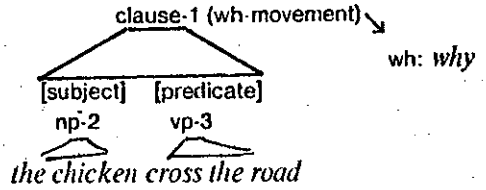
(<abstracting-relation> $?loc (shrdlu :put :block6 $?loc))

4.2 The basic analysis

To handle the basic properties of wh-constructions we need (1) to relate the position of the wh-phrase to that of the rest of the constituents of the clause, and (2) to have some way to create the gap.

i. Positioning the wh-phrase

All but the last three of the list of wh-constructions above are clauses with an initial "wh-phrase". Given the 2-color hypotheses that function words are to originate directly from the grammar rather than from a slot, one might initially imagine that the same should be true for the interogatory or relative pronouns of these constructions. That is, there might be a "*WH*" feature (or perhaps a HOOK) on a clause NODE (as below) and the pronoun printed by [wh enter-node].



This analysis, however, can not be maintained because the wh-phrase is often multi-word and can even embed pronouns—properties which in other cases have been implemented using regular constituent SLOTs.

"How many red peppers did you put in that sauce?"
"How many of them are fresh?"

More importantly, by using a feature on an already realized main clause, the "wh as feature" analysis implies that, at the message-level, these constructions stand in the same relation to the main clause as an adverb or an adjunct, which seems plainly inappropriate.

Instead, the constituent structure analysis that has been adopted draws directly from the notion of the wh-construction originating in a "two chunk" message-level relation. The wh-phrase and the main clause are taken to derive from the abstracted item and the matrix respectively. They are introduced into the tree at the same level as the only immediate constituents of a complex clause using a constituent-schema (named wh-movement) with two slot-names: fronted and matrix.[20]

Interpreted psycholinguistically, this decision to maintain two distinct chunks in the constituent structure predicts that a person should be able to begin one of these constructions and utter the entire wh-phrase without having to complete their decision about what main clause to use, which is in agreement with intuition.

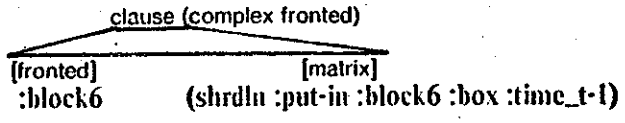
To simplify the rest of this initial section, it will be convenient to use topicalization as the example because it involves none of the complicating grammatical side-issues such as case-agreement or subject-verb inversion. We can start from the point where the CHOICE of

20. This is probably equivalent to the category "S-bar" of X-bar theory [jackendoff_x_bar]; however, I hesitate to use that terminology before doing more analysis and comparison of the systems.

topicalization has been made, i.e. with the CHOICE-APPLICATION:

(topicalize ':block6 '(shrdlu :putin :block6 :box :t-1))[21]

which becomes:



ii. Creating the gap

According to this analysis, the abstracted item appears twice in the message-level source for a wh-construction. Once by itself to identify which message element is being abstracted, and (at least) once in the expression it is abstracted from (the "matrix"). The problem of creating the gap is thus to arrange that the textually first occurance of the abstracted message element within the realization of the matrix is either suppressed (i.e. realized as a TRACE) or forced to be a resumptive pronoun, according to the grammar of the particular construction.

We have seen in the analysis of embedded clauses how a TRANSFORMATION could be used to suppress or modify the [subject] of the subordinated clause. TRANSFORMATIONS do their work at the moment a construction is first laid down as constituent structure, either by changing the mapping of arguments to SLOTS or by editing the PHRASE to remove a SLOT altogether. They therefore depend for their effectiveness on having all of the relevant item's visible at the time that the TRANSFORMATION is triggered. This property makes them unusable as creators of gaps because the element to be suppressed would have to be one of the arguments to the CHOICE. Instead, it will be a subelement (possibly many times removed) of the matrix and thus quite invisible when the clause (fronted) is created. (To prove the point that the gap may appear at an arbitrary distance from the initial wh-phrase, linguists will "construct" clauses like: *Which movie did the critics suggest that the audiences would never believe that the producers expected /.../ ___ would have believable special effects?* These "extensible" clauses, while implausible, are quite grammatical.)

To create the gap, we need a device that will react to the first instance of the abstracted element within the [main-clause], wherever it may appear. One candidate is pronominalization routine (section pronominal_subsequent_reference_n.y.w.). As part of the realization procedure, it already tests every message element as soon as the controller reaches it. A special case could easily be added which noticed if the previous instance of the element was realized as a wh-phrase.

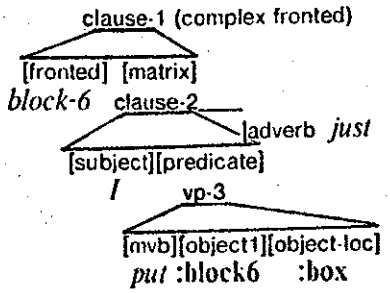
21. This will be realized as 8Block-6 I just put into the box.

Producing a gap would then be a grammar-determined alternative to using a pronoun. The problem with using a special case of pronominalization to create gaps is that it is too inspecific. As we will see, there are syntactic contexts within the matrix which require the use of an interrogative pronoun rather than a gap, or which are insensitive to the influence of wh-movement altogether. Detecting these cases in the context of the pronominalization routine would require setting up extensive contextual checks that have no counter-parts in the normal pronominalization heuristics.

A special-purpose routine Instead, a special-purpose segment wh-segment is added to the realization procedure as a third option, considered before pronominalization or "the main stream". If this option is taken, (i.e. if the ELMT-INSTANCE is another instance of the message element that was abstracted and positioned in [fronted]) then the ELMT-INSTANCE is realized as a TRACE (or whatever, according to the grammar of the construction involved). The necessary sensitivity to context is implemented via two CONTROLLER-VARIABLES: *wh-element* and *wh-trigger*, which are recursively bound by all NODEs with the category-feature fronted. (I.e. whenever such a NODE is entered, the earlier values of the variables are saved and then restored when that NODE is left; see pg.<shallow_binding_n.y.w.>.) They are also bound by the SLOTs [fronted] and [subject] as discussed later.

The value of *wh-element*[22] is the current abstracted message element. *Wh-trigger* on the other hand will have one of the values: *wh-do-nothing*, *wh-trace*, *wh-relative-pronoun*, or *wh-interrogative-pronoun*. It is the flag that tells the wh-segment what action it should take each time an instance of the "*wh-element*" is passed to it for realization. The snapshot below shows the topicalization example just at the moment that the gap is to be created.

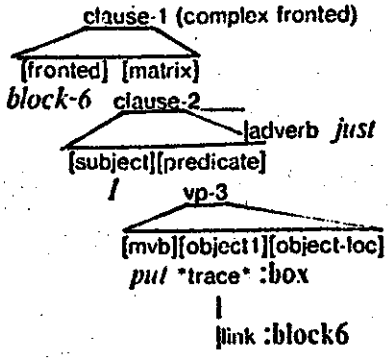
22. A controller-variable is used here rather than a hook because a "controller-variable") can be accessed freely without having to know where it is bound while a hook is only accessed with respect to the node is associated with. Using a hook would still require having a controller-variable that was dedicated to pointing out "the current clause(wh-movement)" and this category of node has not been found to be important enough in the grammar to merit it.



wh-element = :block6
wh-trigger = wh-trace

"Block-6 I just put..."

When the controller passes this instance of :block6 to the realization procedure, the fact that *wh-trigger* has a value activates the wh-segment, which proceeds to compare the *current-instance* being realized with the *wh-element*. If these were not the same-msg-elmt (pg.<same_msg_elmt_n.y.w.>), then we would go on to the procedure's other segments. However, in this case they are the same, and wh-segment decides what realization decision to make by dispatching on the current value of *wh-trigger*. As this example is a topicalization construction, that value is *wh-trace*, and the CHOICE *null-word* is made accordingly. In addition, *wh-trigger*, having served its function, is "turned off" by being set to *wh-do-nothing*. Subsequently, when the wh-segment sees this value, it will know that the gap has already been made and the realization decision should be made by the subsequent reference routine. Below is topicalization example just after this operation.



wh-element = :block6
wh-trigger = *action-completed*

"Block-6 I just put ____..."

Summary and preview At the message-level, wh-constructions are descriptions created by abstracting one element from a propositional matrix; they become two slot constituents: abstracted element followed by its matrix; the matrix is then grammatically marked by being realized with a "gap" where the element would have appeared.

This basic picture must now be sharpened and complicated. We will begin by looking in detail at the kind of information that the speaker must supply in order to correctly specify the wh-phrase in the general case; this will motivate an indexing scheme by which the position of the abstracted element within the matrix can be described. We will then see how details in the "gap-creating" apparatus are varied in order to produce particular wh-constructions, using the indexing scheme to advantage in order to avoid violations of the so-called "island-constraints". Along the way we will consider how this construction interacts with thematic and clause-embedding

WH-segment

same-msg-elmt ?
(wh-element, current-instance)
Yes

Cases of WH-trigger

if *wh-do-nothing* — go on to subsequent reference

if *wh-interrogative-pronoun* — select an interrogative pronoun and return it

if *wh-trace* — construct a trace for the elmt-instance and return it

if *wh-relative-pronoun* — make and execute the grammar-decision to determine

transformations.

4.3 Properties of the wh-phrase

In "proper" English one uses the oblique form of the relative pronoun whenever the gap is positioned in an oblique context:

"*Whom do you wish to speak to ___?*"

thus implying that the position of the gap within the matrix can be known before the controller even enters the [matrix] slot. The only established way of accounting for this "prescience" of the wh-phrase is the literal movement analysis from transformational-generative grammar (see footnote # #); however the linguistic component cannot use it. [[expand on why??]]

Of course, this particular usage is disappearing in modern English[23] and one could imagine designing the grammar so as to ignore it entirely and use only the nominative form. However in other languages it remains very strong. (In Russian for example, the relative pronoun is obligatory and must be in the same case as the gap across the entire declentional paradigm.) Furthermore, we have other evidence that some kind of "pre-planning" of the matrix can be required. The wh-phrase is often a prepositional phrase "extracted" from the matrix:

"With whom do you wish to speak?"
"That is the kind of nonsense up with which I will not put!" (W. Churchill)

Then there is the choice of the wording of the wh-phrase, which may depend on what realization CHOICE is made for the clause containing the gap.

"Where does the jump arc from S/NEG go to?"
"What node does the jump arc from S/NEG lead to?"

The question is whether this kind of information can be determined without a full-scale realization and buffering of the matrix proposition as a finished surface structure—something that would be anathema for the incrementality{-ness} of the linguistic component. This will be answered in the affirmative, but not without placing requirements on the message-level representation.

We will first look at the kind of information that we need to determine just the content of a wh-phrase, and then at an example of a question being planned by a speaker who uses an adequate representation. That example will take us into the question of "prescience" and one way to find out what we need to know about the context of the gap with only a minimum of actual lookahead.

---

23. This is particularly true in colloquial speech. There is an interesting discussion in [linde_there_agreement] suggesting that this phenomena may be related to the also colloquial habit of always using singular number agreement in "there-insertion" constructions (pg.<there_insertion_n.y.w.>). Both may be tactics that allow the speaker to begin speaking without having to plan the remainder of the clause—an ability which could be readily modeled by a conjunction of this theory of production with a model of resource limitations.

i. The need for an adequate message-level representation

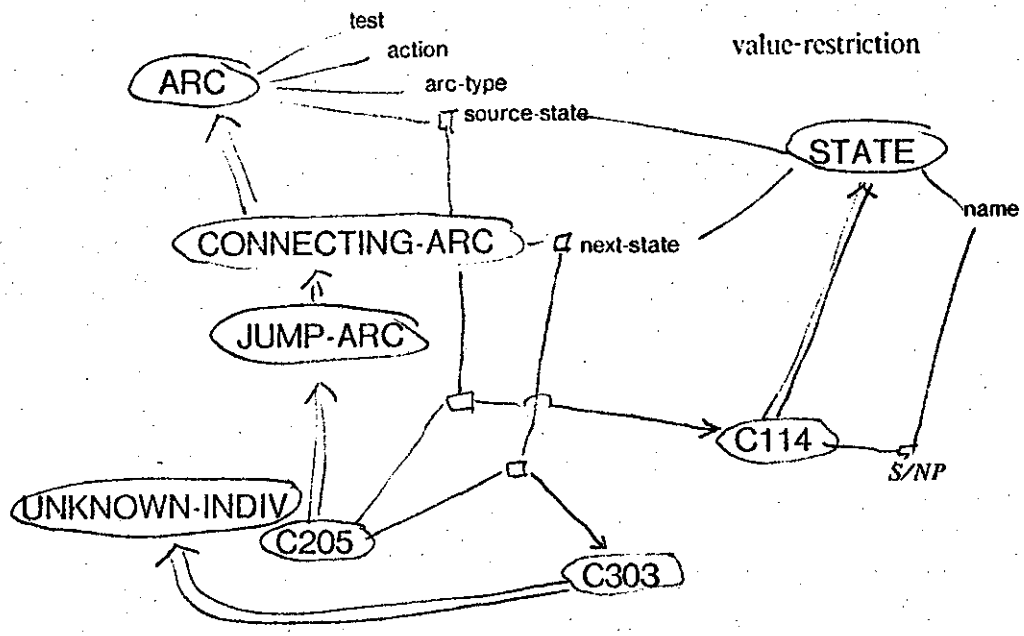
Some kind of description of the generic properties of the abstracted element must be available if wh-phrases are to be properly constructed. Even the simplest phrases—solitary interrogative pronouns—require knowing the element's number, animacy, and/or semantic case. As a consequence, representations that use only uninterpreted variables to denote an abstraction (such as MICRO-PLANNER) are inadequate. They can be used only if supplementary descriptions are included *ad hoc* in the linguistic component's dictionary. Thus in order to use, say, (question $?obj (location $?obj :box)) as the source for *what is in the box?*, the dictionary designer would have to make an explicit link between the message-level token "$?obj" and the interrogative pronoun *what*, and yet another link would be required to be able to say *what blocks* instead.

Just such links were used to augment the representation employed in the logic domain. There, predicate calculus variables were used as the basis of "generic" descriptions, e.g. "*...anyone who...*", "*...all those who...*", or "*...a thing which is...*". A set of features just sufficient to distinguish the different English pronouns (e.g. *human, animate, male, singular*) was created and attached to the individual variables as constants that were accessible to the common dictionary entry variable-entry. (This is described in the appendix pg.<ad_hoc_features_on_variables>.) As is the wont of *ad hoc* techniques, a baroque system was needed to manage these features dynamically in an effort to make up for the fundamental vagueness of standardly quantified variables when compared with English pronouns (see section VII.B.2.1).

On the other hand, if some description of $?obj's generic properties or of the range of possible candiates is available as a natural part of the message representation (or available on demand from the expert program), then the wh-phrase can be constructed from that description directly, without adding anything to the dictionary. Modern, "frame-based" knowledge representations can supply just that kind of information through their annotations of "defaults".

An example from KLONE The accompanying figure shows an example of these defaults as they appear in KLONE. (See page <klone_syntax_n.y.w.> for a description of KLONE syntax.) Imagine that we have a program that builds ATNs according to the specifications of its human user. The figure reflects the relevant parts of the program's knowledge state after it has asked *Are there any jump-arcs leaving S/NP?* and received the (uncharitable) reply: *Yes.*

As a result of the user's instruction, a new individual concept has been created, C205, to represent the set of jump arcs leaving from the state S/NP. The KLONE notation distinguishes what we know about C205 as a specific individual (namely that it has a source-state: C114, whose *name* is *S/NP*) from what we can deduce about it because of the way it has been described, i.e. By warrant of being an "ARC", C205 must have a test, an action, and an arc-type, and, by warrant of being a "CONNECTING-ARC", it must have a next-state (the crux of the following example). We do not know the value of C205's next-state (it is represented as an *unknown*



individual), however, we do know that it must be a STATE, since that is the value-restriction indicated by the generic concept of CONNECTING-ARC.

Aspects of the dictionary for KLONE (an aside) Given the descriptive mechanisms of KLONE, there is a straight-forward procedure for describing anonomous individuals like the one denoted by C205. We begin with a reference to the concept it is an instance of ("*a jump arc*") and then add restricting phrases according to the relations it specifically participates in (e.g. "*...leaving from S/NP*"). Similarly, we can describe unknown individuals in terms of the value restrictions they are known to satisfy and an appropriate relation. The step from descriptions to questions is very short since, given the ability, say, to form the English phrase "*a state*" from the generic concept state, it is a simple matter to substitute the interrogative "*what*" for the indefinate determiner "*a*" and form the wh-phrase "*what state*".

The ENTRYs for generic concepts and their roles are "inherited" by the generic and individual concepts that specialize them. The generic JUMP-ARC, for example, has a "local" ENTRY to contribute the classifier *jump*, and inherits ENTRYs (actually DECISIONs) from CONNECTING-ARC, ARC, and all of their attached generic roles. Generally speaking, the ENTRYs for generic concepts contribute open class vocabulary (often including sets of synomyms) and define what combinations of relations can be used in descriptions. Thus staying strictly at the generic level, one can say:

"A jump arc connects a source state and a next state."
"Jump arcs go from source states to next states."
"A jump arc (from a source state) leads to some next state."

To then specialize a generic ENTRY so as to describe a given individual concept, one uses the same set of CHOICEs and simply substitutes specific descriptions of the individual and its roles for the equivalent generic descriptions. "Mixed" versions are also possible of course; e.g.

"C205 *from* C114 *lead(s) to* (v/r C303 STATE)."

becomes:

"*The jump arcs from S/NP lead to particular next states.*"

A phrase like "*the jump arcs from S/NP*" is produced by a "shopping-list" ENTRY (pg.<shopping_list_style_entrys_n.y.w.>) which has been directed to produce a noun phrase using a specific list of properties; e.g.

```
(Describe 'C205
          '((C205 specializes JUMP-ARC)      ;source of "jump arc"
            (type C205 generic)              ;source of "the ...s"
            (source-state C205 C303)))       ;source of "...from S/NP"
```

At the speaker's option, the description of objects the message mentions may either be explicitly planned (as C205 is here) or left to the discretion of default ENTRYs.

An example of a planned question Our hypothetical ATN building program will want its user to provide a value for the next-state role of C205; this will lead it to asking the user the obvious question, the construction of which we will look at in considerable detail in the succeeding subsections. First of all it assembles the semantic basis of the question:

(1) the individual C303 which it wants the user to identify, and

(2) its chosen description of C303—let us say:

```
(describe 'C303
          '((value-restriction C303 STATE)
            (next-state C205 C303))
```

These decisions made, the question-formation task is taken over by the linguistic component in the form of the CHOICE *wh-question*[24] applied to the two message elements. This is the routine that is responsible for constructing the initial two SLOT wh-movement type node and, in particular, for determining the form of the fronted wh-element.

---

24. As opposed to *polar-question* (*"Are there any arcs leaving S/NP?"*) or *tag-question* (*"There aren't any arcs leaving S/NP, are there?"*).

ii. An indexing scheme to facilitate 'lookahead'

To form the wh-phrase correctly, wh-question must know where and how the abstracted element (C303) would be realized within the matrix. To find this out either it can guess (e.g. always say *who*; see footnote 23) or it can lookahead into the matrix, find the message element that contains C303, make realization decision for that element (symbolically), and look at the result.

Explicit lookahead is not as awkward as it may seem at first, *provided that the message-level representation includes an index from objects to the relations that mention them.* Such an index is trivial to compile if, as in this case of KLONE, messages are constructed by explicitly combining *kernal propositions* according to a fixed relational vocabulary: each composition operation includes a step that incrementally augments the index. The index is a table with one entry for each referring object mentioned in the message.[25] Each entry has two fields: the first is the message element controlling the realization of that object, and the second is a trace describing how that element is embedded within the message as a whole. In the case of the present example, the first field contains the entire matrix relation (i.e. "(and (value-restriction...)(next-state...))") and the second is empty since the that relation is not contained in any other. [[forward pointer to a more involved example]]

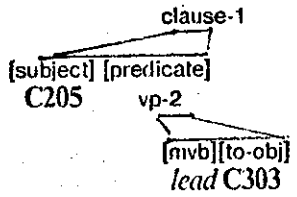
Making the realization decision without producing a realization When we "lookahead" to make the realization decision, we do not want to actually build constituent structure. (Since the controller is not present, there would be no place to position the constituent structure after is had been built.) Instead, the realization procedure is only taken as far as the selection of the CHOICE (including any TRANSFORMATIONS), and this result preserved by the device of an EARLY-INSTANCE. [[[expand??]]]

The realization of this matrix is straight-forward: the message-level relation is and—a relation in the interlingua which instructs the linguistic component to combine its arguments in such a way that all of their information is expressed, but which leaves the decision on just how that is to be done to general heuristics based on its arguments' linguistic properties. In this regard, the first argument to the and is "a description of C303" and the second "a proposition involving C303". There are several conventional ways to realize this combination of arguments: One can build a noun phrase ("A <description> that <proposition>"); one can use the noun phrase to build an existential ("There is a <description> that <proposition>"); or one can build a clause from the proposition, embedding the description with it.

---

25. At this writing, this index is used only for wh-movement and several, rough-cut discourse heuristics; consequently, maintaining an index for *all* of the references in a message is probably wasteful. On the other hand, it is presently simpler at the message level to index all objects indiscriminantly than to have the foresight to know which particular one we will want to know about later. Subsequent, more deliberate speakers will doubtless be able to construct more efficient tables.

In the present instance, we make the realization decision knowing that our intention is to use the result as the basis of a question; this rules out the first CHOICE automatically (i.e. questions are based on clauses). The second is reserved for circumstances that call for focussing on the np's existance, which this is not; this leaves the third CHOICE ("embedding the description within the proposition"). If the second choice had been made that would have determined the linguistic context of the gap sufficiently and we could have stopped our lookahead at that point. Instead we must continue on and symbolically realize (next-state C205 C303) (the embedded proposition). For brevity let us immediately say that it selects the CHOICE below ("C205 *leads to* C303"). For ease of exposition, the CHOICE is shown as a constituent structure. Remember that it is actually a symbolic description in terms of a phrase and a map.



With C303 now mapped into a known slot the lookahead is finished. Its instigator, the choice wh-question, symbolically examines the realization and extracts the information it needs, namely that C303 would have appeared in an obligue context. Of course, since C303 is non-human, the selection of interrogative pronouns open for it does not distinguish nominative from oblique case (i.e. the only possibilities are "*what*" or "*which*"); thus it might appear that this lookahead could have been dispensed with. However, as C303 would have been the object of the preposition "*to*", the marked argument of the selected verb, "*lead*", English grammar gives us yet another possibility for the wh-phrase: wh-question must now decide whether to "pied-pipe"[26] the preposition up into the wh-phrase or to leave it with the matrix, i.e.

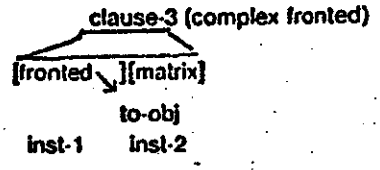
"To what states does S/NP lead?" versus
"What states does S/NP lead to?"

I have no idea of the usage-heuristics that might select between these two alternatives beyond an impression that the preposed version is marked for "high" style. MUMBLE treates them as a synonym set. For this example, let us say that the preposed version is choosen. This means that the SLOT-FEATURE that produces the preposition *to* is added to [fronted] and "removed" from the

26. The term pied-piping was coined by Ross. In this thesis [ross_thesis] pg.109], the covered a larger number of constructions than it does here. There, it included wh-elements in the qualifier phrase of a subject noun phrase, a construction that I analyse as having inhibited any movement and thus handled by a completely different mechanism. An example would be "*Reports* $[_{np}$ *the lettering on the covers of which*]" the government prescribes the height of are a shocking waste of public funds").

matrix (i.e. not added to the slot in the first place).

The final result of wh-question's deliberations—the initial constituent structure for this question—is shown below. The usually implicit ELMT-INSTANCES (inst-1, inst-2) are shown explicitly this time to point out where the message elements have been modified.



inst-1
real-msg-elmt C303
attachments
(describe-as
(value-restriction C303 STATE))

inst-2
real-msg-elmt (and (value-restriction C303 STATE)
(next-state C205 C303))
amended-decisions
(matrix impure-decision-3
preempting-choice *clause_lead-to_wh<=to-obj*

4.4 The specifics of the different wh-constructions

The nine wh-constructions on page 160 can be produced from the same basic constituent-schema by varying the value given to the variable wh-trigger when the controller enters the two toplevel slots: [fronted] and [matrix]. That value can be dictated by using slot-features to tailor those slots at the time they are created; it is rebound by the action of the enter-slot routines of features corresponding to each of the values that wh-trigger is to assume. The names of the features are: i.e. wh-do-nothing, wh-trace, wh-relative-pronoun, wh-interrogative-pronoun.

Each construction is initiated by its own choice, the arguments to which are always the matrix and focus message elements. Each choice will refer to its own phrase for the instantiation of a suitably tailored version of the basic constituent structure, but the lookahead procedure ("determine-wh-phrase") is shared. If the lookahead determines that the wh-phrase should consist of more that just the focus, then the change in the mapping is made before instantiating the phrase. (The lookahead thus appears in the actions property of the choice.)

Determine-wh-phrase and the wh-routine as a whole incorporate a set of grammatical restrictions on the relative location of the "gap". As a result of these restrictions, the [fronted] constituent may be supressed in favor of an "echo" ("*You put it where?!*"), or there may be a forced, radical departure from the matrix's default realization. Discussion of these so-called "constraints on movement" will be postponed until after the individual constructions. (Adjustments will occur only if the message-level representation is sufficiently sophisticated to support them; if it is not, no corrective actions can be taken and there may be grammatical errors in the output text)

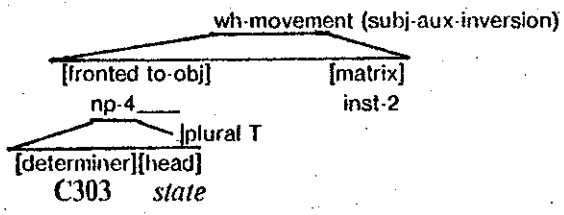
i. Wh-question

fronted: *interrogative-pronoun*
matrix: *trace*

As this is the first wh-construction to be discussed, we will look at it in some detail. The full-scale choice looks like this:

```
(define-choice wh-question (focus matrix)
    phrase (wh-movement (subj-aux-inversion)
                        fronted features (wh-interrogative-pronoun)
                        matrix features (wh-trace))
    map ((focus . (fronted))
         (matrix . (matrix)))
    actions.((determine-wh-phrase focus matrix)) )
```

This is the choice that produced the constituent structure of the earlier example. If we now look at a snapshot of that same tree just a little farther along, we can see how setting wh-trigger to *wh-interrogative-pronoun* will have its effect.



"To... "

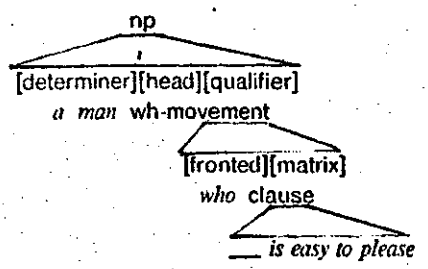
When the controller reaches C303 in [determiner], it passes it off to the realization procedure, where the wh-routine is activated because wh-element and current-instance are the same msg-elmt. The wh-trigger now has its effect: as the diagram on page 113 shows, it indicates than an interrogative pronoun should be selected.

All pronoun selection is done by the same subroutine on the basis of information collected by interface functions to determine number and gender (in this case inanimate and plural), and by determine-wh-phrase to determine the syntactic context within the matrix (in this case oblique). With the fact that we need an interogative pronoun, this is enough to narrow the choice to "*what*" versus "*which*". Which of these to pick is not entirely a grammatical question, and so the choice is made by a grammar-decision, presently on the basis of whether or not the wh-element is a member of a predetermined set: if so, then "*which*" is used, else "*what*".

The slot-feature subj-aux-inversion triggers the "inversion" of subject and verbal auxiliary that characterizes the common form of English questions. This operation is described on page 155.

ii. Relative clause

fronted: *wh-relative-pronoun*
matrix: *wh-trace*



The only substantial difference between between relative clauses and questions is that in a relative clause one often has the option of omitting the relative pronoun. For this reason, the wh-trace case of the wh-routine includes a grammatical-decision with the option to select the null-word instead of a relative pronoun. There are grammatical restrictions on this option however: pied-piping cannot have applied (i.e. one cannot say "*the house in ___ my grandfather lived...* "); and tensed relative clauses that modify the [subject] must always retain their relativizers, as the omission of the relativizer should not create any ambiguity as to what the main verb of the sentence is.[27] These restrictions aside, there are no heuristics established in the literature to govern this decision; and at this writing, the two possibilities have been tied together in MUMBLE as a synonym set, insuring, at least, that they will be treated uniformly in coordinated contexts.

The putative transformation "WHIS-deletion" ("who is") has not been included. It optionally reduces the subject and copular verb of relative clauses, producing "*the girl climbing the tree*" from "*the girl who is climbing the tree*". This omission is for two reasons: (1) The metrical/stylistic differences between these two phrasings are considerable and should be planned at a much higher level than a transformational decision would imply; (2) Given that the speaker and expert program are very likely to have abstract predicates already in their internal ontology, the transformation is not needed to "account" for the existance of the reduced relatives—they can be produced directly. (Indeed, it is more likely that one would want a "WHIS-addition" transformation.)

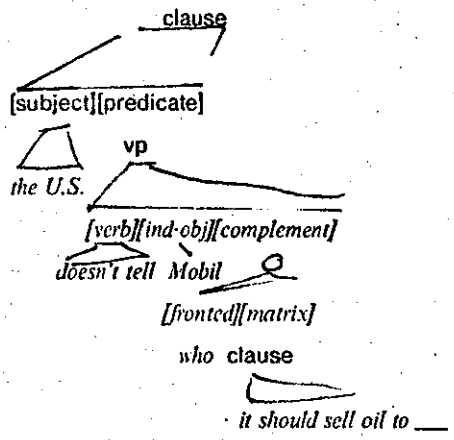
The designer will occasionally want to use a "non-standard" relative pronoun in a situation where the conceptual structure of the domain will not motivate it naturally. For example, in the KLONE-nets-as-objects domain, the entry designer has a very good reason to use "*where*" as the relative pronoun, as in: "*A clause is a phrase where the head role is taken by a verb and the modifier role by a subject.*". This can be accomplished *ad-hoc* by having an entry preempt that determine-wh-phrase decision at the time when the containing construction is instantiated.

---

27. This statement of the constraint as a "filter" on the relativizer decision is entirely analogous to the "surface structure filter" proposed by Chomsky and Lasnick [filters_and_control]. It is interesting to note that the filter which they observed "should" be in the language but was not-- the one that would block the production of, e.g., "*the horse raced past the barn fell*"—is prohibitively difficult to state with the formal devices available to a production grammar of the present design because of the extensive amount of lookahead required.

iii. Headless relative

fronted *wh-interrogative-pronoun*
matrix *wh-trace*



Headless relatives are simply relative clause that stand by themselves rather than qualifying some ("head") noun. They are realized in the same way as a relative clause except that they have fewer possibilities for relative pronouns. Pied-piping is not allowed; this is arranged by having the choice that produces the construction apply a choice-filter to the standard relative clause decision determine-wh-phrase inhibiting it from selecting the pied-piping variations.

Grammatically, headless relatives are interesting because they can appear in syntactic contexts where equi-np-deletion must be applied to their [matrix] constituent. For example in

"The Ayatolla told his ministers who ___ not to sell oil to ___."

the instance of "*his ministers*" that would have appeared as the [subject] of the headless relative was suppressed because the relative is the [complement] of the verb "*told*". This is arranged for by adding the node-feature pass-through-higher-slot-features to the matrix slot as is already done for the slots of conjunctions. In this way, the lower region comes to look like the higher one and the appropriate transformations are triggered.

iv. Topicalization

fronted: *wh-no-action*
matrix: *wh-trace*

Subject to the general constraints on the location of gaps, any constituent of a clause can be topicalized. This way of looking at topicalization brings under its "umbrella" constructions with preposed adverbial phrases and even introductory adjunct clauses.

"Tommorrow morning I'll be asleep."
"By taking an adjacent corner instead of the center, you left yourself open to a fork."

In the usual analysis, topicalization is said to have applied only if the preposed phrase and matrix are spoken without any interveening pause (as would be marked by a comma). The difference here is a guestion of derivation: if the source for preposed phrase is part of the matrix expression

at the message-level, then the construction is effectively topicalization regardless of whether a comma is included or not because once in the [matrix], the suppression of the recurring "topicalized" element will still be required. On the other hand, if the two elements are being brought together just for the purposes of the one speech-act, then the element will not need to be supressed in the matrix because it will not ever appear there at the message-level and the wh-movement aparatus will be superfluous.

v. Left-dislocation

fronted: *wh-do-nothing*
matrix: *wh-do-nothing*

This construction is largely restricted to speech and is included here only for completeness.

vi. Generic relative

This construction (and the three that follow) does not use the standard wh-construction constituent-schema. Instead, it uses the "raw material" of wh-movement, i.e. the features and the grammar-routines, to algorithmically construct a constituent structure appropriate to its needs, in this case, one formed by using a second instance of the focus to construct a generic noun phrase to which a "normal" relative clause is attached. Technically, this is accomplished in MUMBLE by having the choice assemble a special elmt-instance analogous to those created for shopping-list message elements (pg.<entrys_for_shopping_list_msg_elmts>). The example below is from the barber proof (pg.<barber_proof>); here the parameters to generic-relative were the universally quantified variable y as focus (with the features: human and male) and the predicate ¬shaves(y,y); the final realization was "*anyone who doesn't shave himself*".

Elmt-instance
real-msg-elmt y *;for the benefit of the discourse-history*
entry-for generic-pronominal-reference *;produces the "anyone"*
entry-arguments-for y
attachments qualifier clause-1

wh-movement

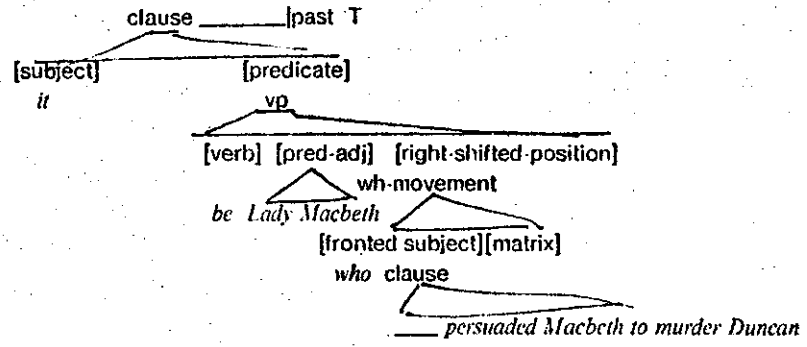
[fronted subject]	[matrix]
y	¬shaves(y,y)

vii. Cleft & Pseudo-cleft

These two constructions are thematic variations on a technique for focussing an arbitrary element of a proposition. They thus involve choices with three parameters: the element, what is to be said about it, and the proposition that describes it. The element and proposition become the focus and matrix of a headless relative (or a generic relative, or a relative clause who relative pronoun is "*that*"), and the headless relative is then made the [subject] of a predicate adjective

construction with "what is to be said about the element" as the predicate adjective. A pseudo-cleft sentence is an extraposed version (pg.<extraposition>) of a cleft sentence.

cleft: "*The person who persuaded Macbeth to murder Duncan was Lady Macbeth.*"
pseudo-cleft: "*It was Lady Macbeth who persuaded Macbeth to murder Duncan.*"



The "fine-tuning" of these constructions (e.g. determining what kind of relative is best, or knowing to avoid stylistically awkward variations) has not been developed in MUMBLE because the present micro-speakers are unable to motivate that kind of focussing.

viii. Tough-movement

In classic transformational-generative analyses, tough-movement is placed in the same transformational family as extraposition. (Opinions are changing however; see [oehrle_li_10_4]).

unmarked: "*Pleasing John is easy.*"
extraposed: "*It is easy to please John.*"
tough-movement: "*John is easy to please.*"

In the present analysis however, the transformations cannot be in the same family because their sources vary in the number of message chunks involved. Here because the message is always realized top-down and left-to-right, a transformationally derived subject must always be an independent chunk at the same level in the message as the rest of the clause (i.e. the same level as the chunks for "*easy*" and "*to please X*") because if the "subject" originated inside a predicate chuck, it could never be noticed in time to move it to [subject] position.

MUMBLE presently has two versions of tough-movement: one expecting two parameters ("*John*" and "*easy to please John*") and the other expecting three ("*John*", "*easy*", and "*to please John*"). The choice for the second is shown below. The grammatical devices implementing wh-movement are impressed upon the basic predicate adjective construction by adding features: the

clause is labeled wh-construction in order to set up the two controller-variables;[28] and the [complement] is labeled wh-trace. All of that is expressed as shown below:

```
(define-choice tough-movement-3 (s a c)
    phrase (basic-clause (wh-construction)
                predicate (vp-pred-adj ()
                                pred-adj (basic-adjective-phrase ()
                                                complement features (wh-trace) )))
    map ((s . (subject))
         (a . (predicate pred-adj adjective))
         (c . (predicate pred-adj adj-complement))
         (a . (*hook* basic-clause wh-focus))))
```

4.5 Planning by the speaker: obeying 'island constraints'

In his seminal thesis *Constraints on Variables in Syntax* [ross_thesis], Haj Ross identified four syntactic configurations[29] that he described as "islands": regions that are grammatically "insulated" from the rest of the sentence, and in particular are opaque to wh-movement.

Complex Noun Phrases:
"*Sigfried killed the dragon that guarded the treasure of the ring.*"
"* *What treasure did Sigfried kill the dragon that guarded ___?*"

Coordinate Structures:
"*Arthur rewarded Gawain and Perceval for their quests.*"
"* *Who did Arthur reward ___ and Perceval for their quests.*"

Sentential Subjects:
"*Rescuing fair maidens every day requires stamina.*"
"* *Who does rescuing ___ every day require stamina?*"

NP's that are embedded as the leftmost branches of other NP's:
"*The flowers were wilted by the dragon's fiery breath.*"
"**...the dragon whose the flowers were wilted by ___ fiery breath...*"[30]

The import of these constraints for production is very different from what it is for linguistic competence. In production, one does not have the option of marking a sentence "ungrammatical" and starting over; instead, the syntactic configurations identified by constraints like these must somehow be avoided in the first place: grammar is a prior constraint—not a postfacto censor. The constraints on movement are in this way no different that other principles of grammar, only less

---

28. The mechanism used in MUMBLE to "tell" the wh-element what value it is to have in these cases is more than a little ugly. A hook is attached to the clause node with the desired value; a default-decision then "activates" the variable once the controller reaches that node. This awkwardness is largely due to not allowing controller-variables to be set directly by the evaluation of choices; should it continue to crop up, this awkwardness would be a good motive for reanalyzing the original motivations behind that detail of the design.

29. Since their original description, Ross's constraints have been reinterpreted and redescribed many times in response to empirical observations and changes in linguistic fashion. It would be besides the point to discuss the merits of these changes or even to review them since the planning problem that the constraints pose for production remains the same regardless of the details of the constraints' formulation in transformational terms.

30. The effect of this constraint is to require that the entire containing NP be pied-piped to [fronted] along with the focus, i.e. "*...the dragon whose fiery breath the flowers were wilted by*".

obvious.

Logically, this "avoidance" could occur either at the level of message assembly (we never think to say anything that would be a violation) or at the level of message realization (messages that would lead to violations are detected and treated specially). Of the two, the first is certainly the simplest from the linguistic component's point of view, since if it is true, there is no need for linguistic detection and transformation facilities. However, it is unlikely that this would happen just by serendipity. Rather, it would mean that complex-NP's, coordinate structure, sentential subjects, and left-recursive NP's are somehow directly selected by whatever non-linguistic prinicples of communication speakers obey when they decide what to say.

On the other hand, if the constraints do not have a cognitive basis (and we are not presently in a position to know either way), then they are just further "channel characteristics" that the linguistic component must impose on messages as they are realized, just as agreement and word order are imposed. Should this be the case, the question of how much search the linguistic component must undertake in order to notice potential violations in time to forestall them becomes very important if a linear-time process is to be maintained. The rest of this section will describe a combination of message indexing and grammatical alternatives to wh-*movement* that, for the most sophisticated micro-speakers, appears to accomodate the constraints while incurring only a constant overhead.

Right-thinking Since the linguistic component selects constructions only when they are motivated by the message, we can reinterpret this question of constraint violations as a question of the speakers motivations: "what would motivate the speaker to want to say something that would violate a movement constraint?". If the necessary combination of motivations never exists, then messages that lead to violations will never be produced and there is no problem. Consider the famous overly embedded sentence:

"*The rat the cat the dog chased killed ate the cheese.*"

If we assume that the motive for making a message element the [subject] of its clause is that we want to say something about it, then this sentence is the result of the bizzare plan to use a fact focussed on the cat in order to identify the rat (and similarly for the cat, to identify her by means of a fact focussed on the dog). A more sensible plan would insist that restrictive relative clauses must be focussed on the objects they qualify. This would yield the far more understandable formulation:[31]

"*The rat that was killed by the cat that was chased by the dog ate the cheese.*"

---

31. The remaining awkwardness is due to embedding one description directly within another, in effect distracting the audience from the fact that our subject is really the rat—a side-effect that could be predicted from the fact that speech is produced as a sequential stream. If a description of the cat is a necessity, then perhaps a reasonable reformulation would be "*The rat that ate the cheese was the one that was killed by the cat. That cat, in turn, was the one that was chased by the dog.*

Violations of the movement constraints appear to have a similar analysis; certainly it is the case that any sentence that violates the constraints will invariably have an alternative formulation where the role of the wh-element is markedly different. Thus for:

"** What treasure did Sigfried kill the dragon that guarded?*"

we can instead say:

"*What treasure was guarded by a dragon that Sigfried killed?*"

or for:

"** Who did Arthur reward and Perceval for their quests?*"

we can say:

"*Who besides Perceval did Arthur reward for their quests?*"

When complex-NP is violated, it is because the wh-element's role in the matrix was extremely minor, i.e. part of a qualifying description. Similarly in violations of coordinate structure, the wh-element is part of a list in the matrix and thus indistinguishable from its fellows.

Unfortunately however, conjectures about the motives of human speakers are difficult to substantiate (except perhaps for linguists who are telepaths), and those programmed speakers that do exist are as yet too simple to be a base for convincing synthetic experiments. I can therefore only observe that there appear to be no compeling reasons why a speaker should ever want to describe a wh-element in so indirect a manner that a violation of a movement constraint would occur. If that is in fact the case, then the linguistic component need not worry about them.

Dictionary-level alternatives While it may well be that a speaker would never plan to violate the constraints, it is more typically the case (at least with the micro-speakers) that the speaker does not plan at all. Instead there is only a top-level directive to, e.g., "describe S/NP" using the facts "(next-state C205 S/DCL)" and "(source-state C205 S/NP)". It then falls on the shoulders of the dictionary to try to realize those facts in such a way that the constraints are not violated.

If the contents of the [matrix] are really a proposition, i.e. if their internal structure is invisible to the both its entry and the linguistic component, then there can be problems. Either the entry can guess that there will be no violation and go through with its default realization, taking their chances with grammar-level alternatives described below (this yields an apposative: "*...S/NP, the jump arc from it leads to S/DCL*"—not as pleasing, in most cases, as a relative clause would have been), or it can step outside the linguistic component's design constraints and perform a thorough forward simulation and backup (effectively what is sometimes done in the logic domain; see section reasoning_about_possible_choices_n.y.w.).

On the other hand, if instead of as an opaque proposition, the matrix is presented as a network of kernal propositions that some entry would have to "flatten out" anyway, then it is possible to in effect embed a respect for the movement constraints directly into the flattening procedure. In this particular case of C205 this would happen as follows.

@@@@First of all because this description is to function as the [qualifier] of an noun phrase, there is an associated transformational-family to screen the entry's choices. It will filter out any choice that violates a movement constraint provided, of course, that the potential violation is visible to it. The conjunction-entry (developed originally for the logic domain) consults the object/position index and determines that because the two propositions have a common element as their default [subject]'s, they may be merged into one clause by making one of them a relative clause. If has, however, no *a priori* preference for which to merge.

This is the point where the transformational-family steps in. It knows from the position of this realization in the tree that movement constraints apply to S/NP, and from the index that S/NP is a subelement of (source-state C205 S/NP). This is sufficient information to know to filter out any choice that positions that proposition in an island.

Computationally, the index is used as a means of avoiding a scansion of the message during the lingustic processing in preference to scanning while the message is assembled and remembering the results. In domains where the message is compiled by the aglomeration of relatively small propositions from the expert, there is almost no redundant scansion required, since the information can be compiled at a one-time constant additive cost by being piggy-backed onto the message-assembly process.

Alternatives at the grammatical-level If the speaker's representation is unable to support an object/position index, is there any recourse available to the linguistic component short of a dictionary-based simulation? What would happen if the speaker were "taken on faith" and expected to instigate only occasional violations? The answer is that while the violations cannot be forestalled, it can be arranged that the errors that result are the same as the ones made by people. For example from errors collected by David Fay [fay] we have: "*Look at those clouds are moving how fast*" and "*Linda, do you talk on the phone with which ear?*", and from my own collection: "*I have all this memorabilia that I have to decide what to do with it*".

The key to this arangement was aluded to in the initial discussion of the mechanism for creating gaps, namely the association of wh-trigger-controlling constituent-structure-labels to the syntactic configurations that define islands. For versimilitude, the kind of enter-slot routines with the labels are made conditional on the type of wh-movement going on: questions and headless relatives get wh-interrogative-pronoun, and the others wh-do-nothing which has the effect of triggering an ordinary pronoun.

s/np, the jump arc from which leads to s/dcl

A case like Ross's famous "pied-piping" sentences (footnote 172), where the wh-element is overly embedded within the qualifier to a noun phrase, requires the element to be realized as a interrogative pronoun. [[[as in this example]]] whereas A case where the element appears "unexpectedly" in an apposative that happens to preceed its "real" position in the clause calls for

using a pronoun. This example was made by myself in writing this paper:

> "*...counter-parts in* MUMBLE*'s design which, to the extent that the design is independently motivated, ____ may provide an explanation for them.*"

The development of a system of grammar-level alternatives to planned wh-movement is still very under development. Recent research in to wh-phenomena (a very busy area in linguistics at the moment) suggests that among languages, English is unusual for allowing "movement" across such enormous spans of constituent structure. Other languages are far more restrictive, often allowing only the primary constituents of the clause to be moved. Even in English, it may be more reasonable to mark those constituent positions that permit movement through themselves than their to mark those that prohibit it (see [koster_li] and the references he cites). If this is the case and those markings depend on relations that have consistent correlates at the message level, then the transformational technique should work well.

1. Heavy-phrase shift

Some linguists hold that there is a stylistic rule in English and other languages whose function is to increase the intelligibility of a text by causing "heavy" phrases to appear to the right of their "normal" positions [Yngve_model_&_hypothesis][Ross_thesis]. For example, when the direct object of a verb that takes a particle is short, the particle can appear either before or after the object:

"*He saw the matter through.* — *He saw through the matter.*"

But when the direct object is very long, the particle must precede it if the sentence is not to be awkward or even unintelligible: (example from [Yngve_model_&_hypothesis])

"*He saw through the matter that has caused him so much anxiety in former years when he was employed as an efficiency expert by the company.*"

Heavy-phrase shift is also held to be a sometimes source for: extraposition from subject

"*It is not true that there is some barber who shaves all and only those who shave themselves.*"

extraposition of np qualifiers

"*The review just arrived of that book you wanted to read.*"

"*There is usually an issomorphism between the data structures and the desired English texts that a dictionary can be designed to capitalize on.*"

and the order of prepositional phrases in a verb phrase:

"*They dismissed as too costly a plan for the state to build a sidewalk from Dartmouth to Smith.*"

"*A record is kept, by slot,*[1] *of the elements that were formerly embedded there.*"

These constructions are, of course, also used in response to thematic criteria, and it is sometimes difficult to say which is the controlling principle. When more thorough analyses are made, heavy-phrase shift may turn out not be an important factor after all. The point of this section is not to take a stand on that question but rather to show how heavy-phrase shift could be incorporated into the grammar if one wanted it, and what limitations there would be on when and how it applied.

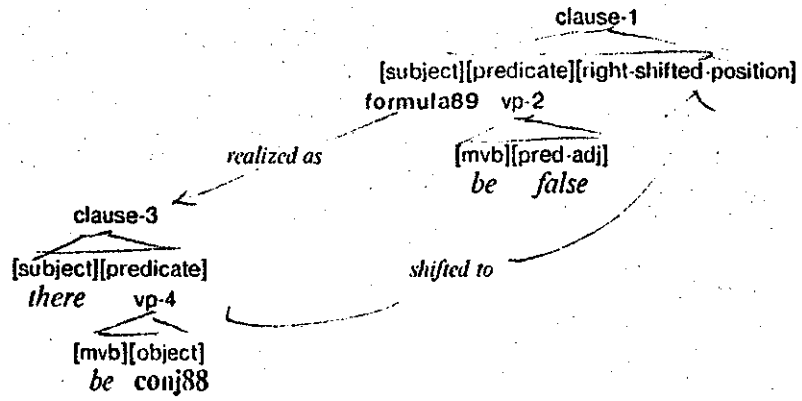
Design options In general, any phenomenon that depends for its trigger on the linguistic properties of a message element can be implemented in either of two ways. One can wait until the controller reaches it within the tree and test its properties after it has been realized (but before the realization has been attached to the tree), or one can anticipate its properties either by looking at

1. A definite "comma intonation", akin to that used for parentheticals and apposatives, is needed in some cases. In a speech-based theory, the need to plan this intonation and its physiological preparation might be a force against the "last-minute decision" analysis of heavy-shift and in favor of planning the decision at least at the level of the verb phrase.

message-level correlates or by actually performing some or all of the linguistic decision-making "in advance" and remembering the results. In the first case, one is certain of what the properties are while in the second, there can be some uncertainty—some decisions (such as pronominalization) cannot be made properly until the entire linguistic context up to the element's position has been fixed. On the other hand, there is a great difference in the lattitude of action permitted in the two cases: waiting "until the last minute" means that only those parts of the tree that are in advance of the controller's position can be affected, everything that came earlier would have already been spoken and could not be changed; anticipating the situation means wider possibilites for actions: the element's own realization can be influenced as well as its position relative to the constituents before it, or an different construction selected altogether.

Part of the choice of implementation for heavy-phrase shift depends on how "heaviness" is measured. Let us assume for the sake of discussion that noun phrase is heavy if it dominates a clause. (This is the criterion of [ross_thesis] pg.28].) The "easiest" way to test for the presence of a relative clause is to wait until the message element in question has actually been realized (but not yet placed in the tree) and to inspect it directly. If, for example, we were using this technique to test whether to do extraposition from subject, we would perform the test as part of the [subject after-realization] GRAMMAR-ROUTINE.[2] The action of this routine is illustrated in the diagram below. The message element in the [subject] has been realized but not yet introduced into the tree. Because the realization is a clause, it is shifted to a newly created latent slot (pg.<dynamically_created_patterns_of_constituents>) with the slot-name right-shifted-position. The "*it*" that then appears to fill the [subject] is actually a directly-printed grammatical reflex of the fact that the [subject] is empty and the CONSTITUENT-SCHEMA is what it is. (The shapshot shows the effect of heavy-np-shift at [subject after-realization] for the last line of the barber proof (pg.<barber_proof>): "*(therefore) it is false: there is no such barber*".)

2. Actually, one would want to put it on a slot-feature, heavy-shift, that the [subject] slot would share with [object1] and with the [qualifiers] of noun phrases. A shift from any of these slots will go to the same position—just after the verb phrase. Since not every [subject] can be extraposed (only those in predicate adjective constructions), this feature would be added to the [subject] by clause-level CONSTITUENT-SCHEMA of the appropriate sort.




```
(define-label heavy-shift
    after-realization
        (cond ((or (equal 'clause (category current-contents))
                   (and (equal 'np (category current-contents))
                        (will-be (get-slot 'qualifiers current-contents)
                                 'clause)) )
;then perform the heavy shift
(set-current-contents nil)
(fillslot current-contents 'right-shifted-position current-clause)))
))) )
```

Alternatively, heaviness could be assessed at the message-level, provided, of course, that the representation for messages made that information easy to extract. For simple entrys, this might mean just reading out their will-be properties directly, but entrys with more contingent behavior could require a simulation (see section reasoning_about_possible_choices_n.y.w.).

Grammatically forced shift In English, it is ungrammatical to simultaniously use an preposed auxiliary and a sentential subject. (This is known as the "internal S condition" [ross_thesis])

"** Did that you became a linguist surprise anyone?*"

As with other "constraints" that linguists have identified, the question for production is to determine what steps can be taken to avoid these constructions (assuming that, in fact, their precursors are actually motivated by speakers). In this case there is a straight-forward "patch" that can be applied should a text ever be in danger of violating this constraint, namely extraposition to the end of the clause. The constraint is thus translated into production terms as an obligatory, grammar-controlled transformation, tested for as part of [subject after-realization].

Planning versus editing Introspective evidence suggests that, at least for humans, it is easier to make a working assumption about how a text should be ordered, produce it, and then examine and edit the results rather than to attempt to predict all of the relevant relations in advance in some abstract planning space. This is particularly true when subtle evaluations of relative heaviness are involved.

Sentence one below shows a "first cut" at a sentence written for this report that was then edited as shown in two.

(1) "*Such constraints are possible because messages are expected to be planned, reflecting the relative importance of the speaker's several goals and their dependencies directly in the structure.*"

(2) "*...reflecting directly in the structure the relative importance of the speaker's several goals and their dependencies.*"

I made the edit in part because of the length of the direct object, but more to avoid the potential ambiguity of the original where the second instance of "*their*" might have been construed as referring to "*dependencies*" rather than to "*messages*" as intended. This ambiguity was one I would never have anticipated before actually producing the utterance, and underscores the subtleties that can go into these decisions in real life.

With some effort, MUMBLE's planning facilities could be extended to predict a problem that far away from the controller. This would require using the object/position index, the could-be properties of the entrys (assuming they were not too contingent), and some kind of presentational buffer that would make it possible to formulate questions about the relative order of projected texts. One might even be able to argue that the, planning not withstanding, MUMBLE would even still function as an indelible process, but incrementality would be much harder to salvage. It is questionable whether one would actually want to implement this kind of extension—it might not be cost effective when compared to *post-facto* editing and restarting. It is intriguing to speculate that there is something about the manner in which language is represented in the human mind that makes that kind of abstract planning so difficult for us.

2. Ellipsis

Certain natural languages constructions permit segments of their text to be optionally omitted, presumably because the periodic structure of the remaining text makes those segments predictable. English has many such elliptical constructions; the first four below have been incorporated into MUMBLE's grammar; the others have not yet been motivated by any of the micro-speakers.

subject reduction: "*He left the party early and ___ went to bed.*"

verb reduction: "*Macbeth murdered Duncan by himself and Banquo and Fleance /!*" ___ by the action of his action of his henchmen."

verb phrase deletion: "*If Mitch has an exam, then Kurt does ___ too.*"

gapping: "*Pobj must be a np, prep ___ a prep, interp ___ a relation, and ppobj ___ a pp.*"

slucing: "*He's seeing someone, but I don't know who ___.*"

comparatives: "*Pansies smell better than violets do ___.*"

comparative elision: "*Sadat is more willing to get along with Begin than Begin ___ with Sadat.*"

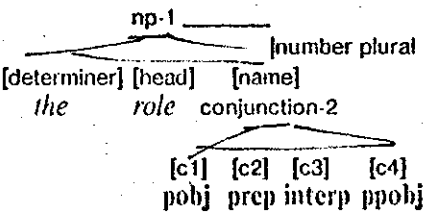
left-peripheral deletion: "*The union leaders met with management on Thursday, and ___ (with) the arbitrator on Friday.*"

What the "proper" message-level sources for these constructions should be is probably a matter of taste and ultimately of what is convenient for the designer of the speaker/expert-program. Either the speaker will prefer to pass complete propositions that the linguistic component will then suppress parts of, or it will prefer passing only the pieces, in effect removing the redundancies at the time the message is assembled (e.g. smell-better(pansies, violets)).[3] Producing ellipsed constructions from specific, already selected elements is quite straight-forward: one simply designs the necessary "piece-wise" CONSTITUENT-SCHEMAS and CHOICEs with the appropriate maps. Consequently, in this section I will only consider cases where the linguistic component must actively "reduce" structures at the message-level.

2.1 Coordinate structure

The periodicity of the text that makes ellipsis possible is usually the result of some kind of conjoined or otherwise coordinated constituent structure. It will be useful then to digress momententarily to describe MUMBLE's treatment of coordinate structure.

Conjunctions are ubiquitous. Every position in constituent structure that can contain a single phrase can contain a conjunction of phrases—all of the same category—and every category of constituents can form conjunctions of any length, limited only by pragmatic considerations. Conjunctions are analyzed here as a their own syntactic category; they can have an arbitrary number of constituents (pg.<schema_with_arbitrary_numbers_of_constituents_n.y.w.>) based on the kernal-slot-name "c", e.g.



3. There is also, of course, "the middle way". For the construction of coordinated referring phrases in the tic-tac-toe domain, I have experimented with passing both the complete propositions and an instruction specifying along what dimension to contrast the two references, e.g. "*Your threat countered mine.*"; see instructions_for_contrast_n.y.w.. Similar tactics could be used to explicitly plan other kinds of ellipsis.

Conjunction NODEs are effectively transparent to grammatical relations:[4] the case of pronouns is inherited, access to HOOKs is passed through, and function words are repeated (subject to the same heuristics that govern category reduction (pg.38). Any active grammatical properties of the SLOT containing the conjunction are applied to each of the conjunction's constituents *after* the properties of the conjunction itself. These effects are brought about by the controller itself because of the feature, *pass-through-higher-slot-features*, that is included on each slot.

For the garden-variety "list" type conjunctions, there is a set of GRAMMAR-ROUTINEs associated with the "c" SLOT-NAMEs which implement the trailing commas and ultimate conjunction. Moreover, the more "interesting" properties of conjunctions, in particular the facilities for managing parallel structure and for triggering ellipsis, are associated with special SLOT-FEATUREs because they are found with CONSTITUENT-SCHEMAs other than just lists; for example one can say:

"Since Mitch has an exam, Kurt must also."
"If Mitch has an exam, (then) Kurt does too."
"Mitch has an exam, so Kurt does too."

The CONTROLLER-VARIABLE *previous-conjunct* is managed by routines attached to the SLOT-FEATURE *coordinated-slot*, one of the intrinsic features of, e.g., [if-slot] and [then-slot] (pg.<definition_of_if_then>). Its value is the RECORD compiled for the NODE in the immediately previous SLOT of the conjunction. (When the controller is in the first SLOT of the conjunction, the value of *previous-conjunct* is nil. This fact can be used as a test to inhibit tests for ellipsis inside that first item.)

2.2 Triggering conditions for ellipsis

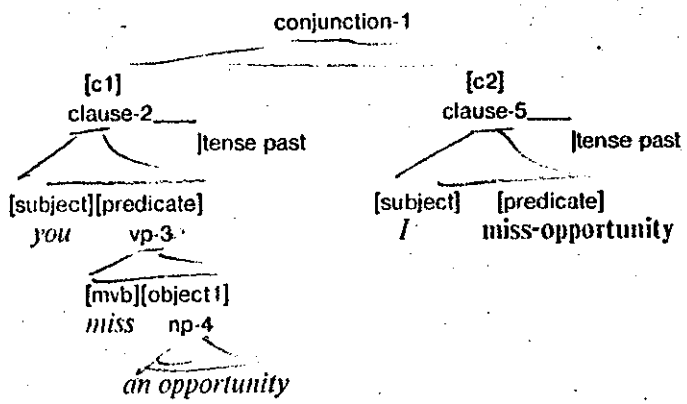
The trigger for ellipsis is a confluence of events that are external to the decision-making of individual ENTRYs. Since semantics plays no part (except to say that two message elements are or are not the same), the responsibility for detecting these events and making the decision to employ ellipsis belongs outside the ENTRYs—the question is whether ellipsis should be a general alternative to pronominalization or only associated with particular structural configurations in the tree.

The pronominalization approach has both good an bad points. Ellipsis is in many ways an anaphoric relation triggered "just" by the potential repetition of a constituent. Also, as the verb phrase deletion ("VPD") examples below show (from [sag_thesis]), the range of structures that permit ellipsis is quite large. Seeding them all with individual grammar-routines to look for possible VPD at different structural locations seems somehow excessive.

4. All [passive tests] against the tree include special checks for the case where a conjunction is involved.

main clause to adjunct:
"*Gwen hit a single after Sandy did ___.*"
deeply embedded subject relative clause to complement:
"*The fact that Bill said she didn't break the window makes me wonder who did ___.*"
One person's question to another's answer:
"*Who hit the home run? Betsy did ___.*"
predicate to self-embedded (!!) relative clause:
"*I kissed everyone you told me I should ___.*"

On the other hand, there are definite limitations to the types of realizations permitted in pronominally triggered VPD. Consider this snap-shot of the tree (from tic-tac-toe).



You missed an opportunity, and I //

If we were to wait on the ellipsis decision until the second instance of miss-opportunity was actually reached, we would have lost the option to use the phrase "*...and so did I*" because the position where the "*so*" would have appeared has been passed, i.e. the only choice available would be "*...and I did (too)*". If both of these are to be "equivalent" grammatical options, then the test must be made earlier, say at [coordinated-slot after-realization].

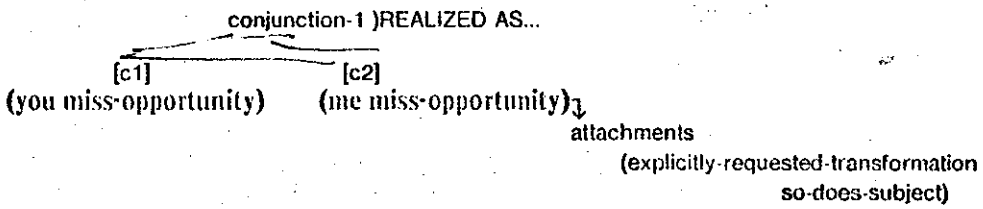
Of course, the two options may *not* be equivalent. Complex sentences with *although* or *but* are readily planned as a unit (see [davey]). If *so* is treated the same way, then there would be an explicit feature on the tree at the level of the SLOT [c2] to implement that specific form of ellipsis. (The pronominal trigger would still apply serendipitously to handle the unplanned cases of VPD; see pronominalizing_predicates_n.y.w. for details.)

2.3 Reduction at different levels

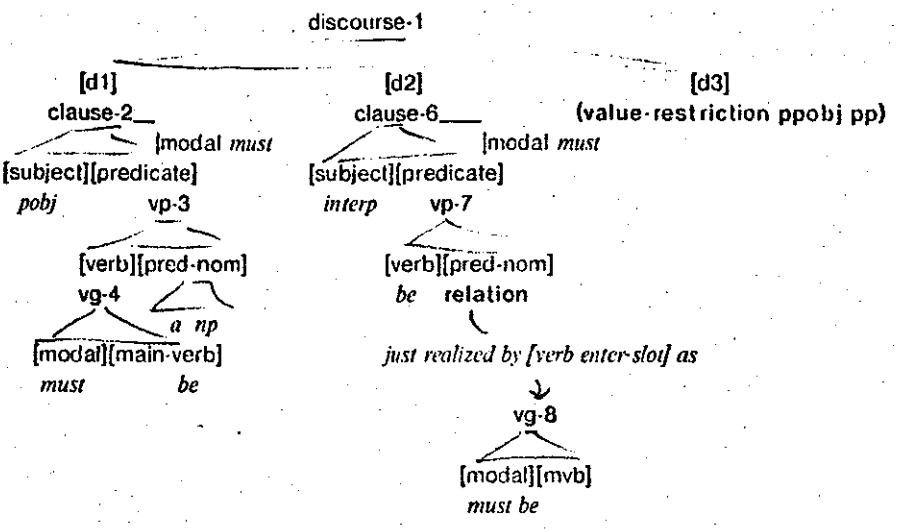
Ellipsis should be detected and implemented at the point where this can be most easily arranged. This has been different for different micro-speakers, according to the representation they employed.

entry-level decision First of all, detection and implementation cannot always be performed together. For the case of the "missed oportunities", the observation that the two predicates are identical and the decision to use the "*...so does*" construction would be made very early before either of the conjoined clauses was actually built. It would be made by the entry for "coincidences"—the relation in the tic-tac-toe domain that is the message-level source for the whole sentence, shown below. (The actual data structure has many more details describing the moves which have been omitted for clarity.) At the time when the decision is made, it is impossible (and not appropriate) to actually implement the ellipsis because the morphemes involved are still latent in message-level structures. Instead, an attachment is added to the elmt-instance that is created for the second conjunct; its efect is to trigger the "*...so does*" TRANSFORMATION when the instance is finally realized.

coincidence ((you miss-opportunity)
(me miss-opportunity)) *MESSAGE-LEVEL*

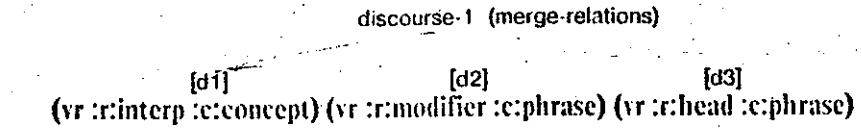


Grammar-level decision *Gapping*, on the other hand, is tested for and implemented at the same time (i.e the last possible moment). Its distribution is more restricted than VPD—list-type conjunctions only—accordingly, it is made part of the GRAMMAR-ROUTINE [mvb enter-slot]. Below is the test for gapping presently used by MUMBLE and a snapshot showing the state of the tree at the time when it would be applied. This snapshot is from the KLONE-nets-as-objects domain, and the test is essentially the same as used for category-reduction in general. (The name "vg-#" stands for "verb group". This category and its automatic construction by the grammar are discussed in section III.2.4.

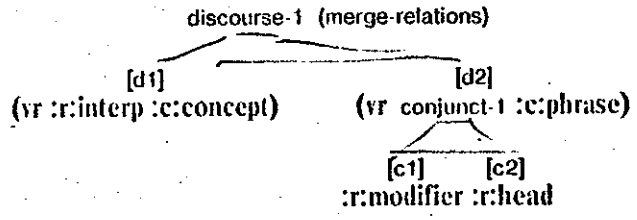



```
(define-grammar-decision do-gapping-if-possible
    condition ((same-contents (verb-group_in_previous-conjunct)  ; vg-4
                              current-contents))  ;the vg just constructed: vg-8
    default (gap) ;do it unless there is a reason not to.
    ((member 'do-not-gap (strategies_used_in_previous-conjunct))
     (do-not-gap))
    ((was-excessively-long previous-conjunct)  ;e.g. included an apposative
     (do-not-gap))
    ((gapping-expressly-blocked)  ;see section <coordinating_alternate_ellipsis_stategies> below
     (do-not-gap)))
```

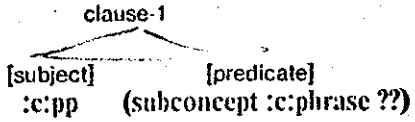
Message-level reduction From the same domain, we have a good example of performing the reductions at the message-level. Consider that speaker's job: to create an English "read out" of a KLONE net. These nets (example on pg.<example_klone_net>) are comprised of arbitrary numbers of very similar relations. Because of this, reductions of the text will always be appropriate, the question is how to organize them. My decision was to make the dictionary-level operations as simple as possible: all that *concept-defining_entry* (pg.<concept_defining_entry>) does, for example, is to collect relations of the same type and make them the constituents of a discourse node. It is then left to a discourse-level GRAMMAR-ROUTINE to scan the relations for reducible patterns. We can see this in the constituent structure below: a snapshot taken just after it was constructed by that entry but before the merger operations were applied. (Note the NODE-FEATURE *merge-relations*: it is its grammar-routine that performs the scanning and reducing. **"vr"** is an abbreviation for value-restriction.)



[Merge-relations enter-slot] will make as large a reduction as is possible without changing the order of the relations. Its output is a new immediate-constituent property for the NODE, with the actual assertions modified as shown below.[5] This eventually becomes the text: "*Its interp role must be a concept, and its modifier role and its head role must be phrases.*".



This practice of constructing *derived* message elements within the linguistic component is an expediency that broadens the range of constructions open to the speaker of conceptually simple expert programs. The KLONE representation language, for example, has no primitives for expressing the abstraction "is a subconcept of X" directly, and thus has no natural source for statements such as: "*PP is one subconcept of phrase and np is another.*" However, these abstractions can be constructed by the dictionary as a by-product of its normal operation. Given the assertion: (subconcept :c:phrase :c:pp), the subconcept-entry builds the NODE below, creating a derived message element by the simple mechanism of duplicating the assertion while replacing :c:phrase with the symbol **??**.



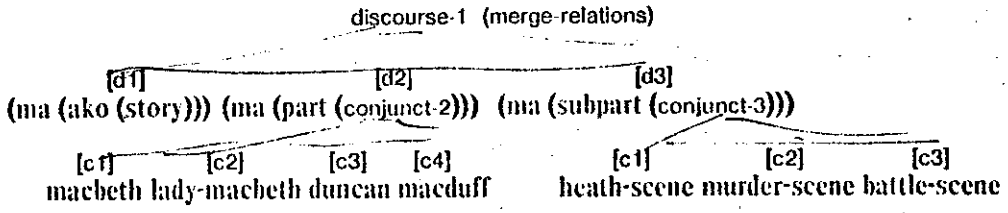
The derived message element is realized by the original ENTRY, which explicitly looks for the **??** symbol and produces the appropriate reduced phrase when it is found (in this case a verb phrase).

Bending the rules for the sake of the speaker In all of the earlier examples, the fundamental criterion for reduction was the presence of two elmt-instances of the same msg-elmt at linguistically significant positions in the tree The different levels corresponded to different times when the linguistic significance of the positions could first be recognized. Linguistic studies (specifically [sag_thesis]) have demonstrated that coherent equality-based rules for ellipsis can be stated only if non-linguistically represented abstract predicates are used as the representation over

5. Since entry for relations treat their arguments as atomic, they will not notice this substitution of a linguistic node for a message element.

which the equalities are defined. Attempts to formulate ellipsis phenomena in terms of the string-matching of morphemes or of p-markers (syntactic trees) will miss significant generalizations (not to mention being incompatible with indelible production procedures).

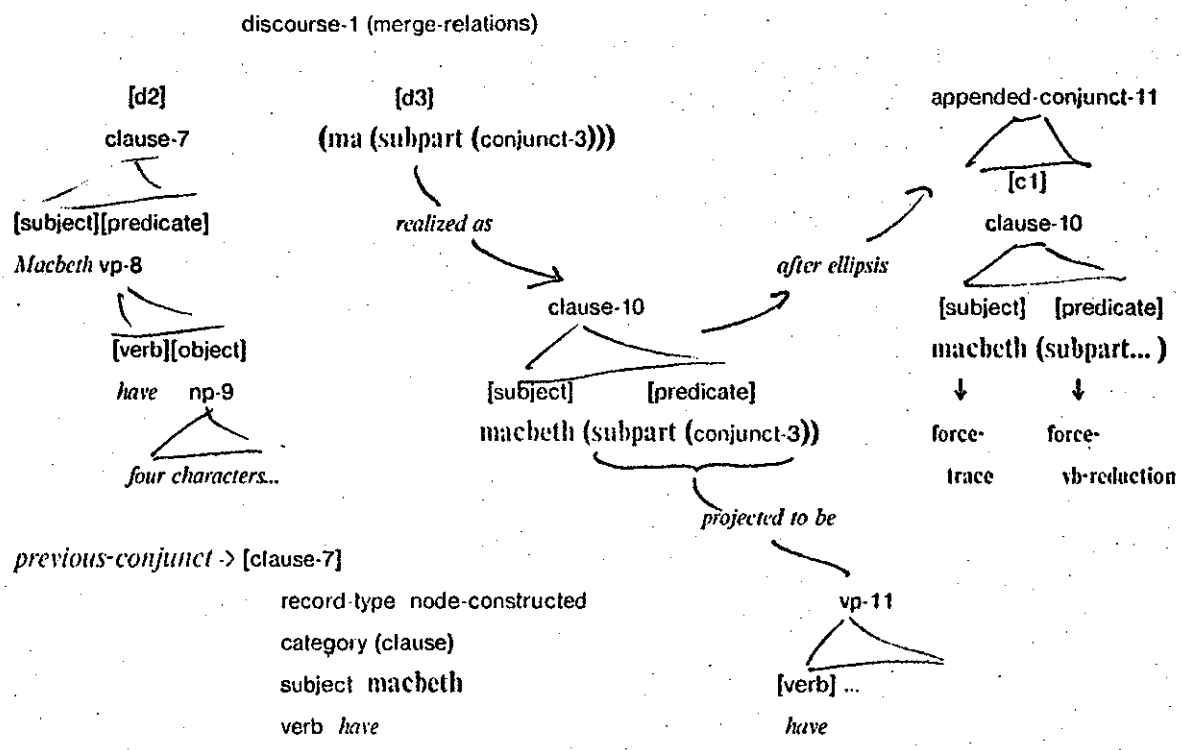
However, as a practical matter, the internal representations used by some speakers for their messages conceal regularities which appear only once the message elements involved have been realized but which should be subjected to ellipsis if the resulting text is to be natural English. A case in point comes from the Macbeth domain: One of the texts from the introduction (repeated below) originated from different message-level relations (i.e. part and subpart) and still underwent subject and verb reduction.



"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.

It turns out that both part and subpart are realized with the verb *have*; however, we can not know that until both realization decisions have been made. Consequently, to perform the reductions we have to use a late-acting GRAMMAR-ROUTINE, specifically [coordinated-slot after-realization]. (In MUMBLE's grammar, the constituent SLOT's of any discourse node with the feature *merge-relations* are all given the feature *coordinated-slot*.)

Doing the tests for ellipsis at the surface-structure level requires considerably more apparatus than doing them at the message-level. The snapshot below shows the tree at the point when the test is made. The new [subject], macbeth is compared with the original (message-level) contents of the [subject] as given in the the RECORD for the *previous-conjunct*; since they are identical, the routine then looks ahead slightly to determine if their [verb]'s will also be the same (see pg.186); in this case they will be, and [coordinated-slot after-realization] performs the subject reduction by marking that instance of macbeth for transformation into a TRACE and marking the [predicate] to force a similar suppression of the verb (discussed below). (The category *adjoined-conjunct* is a device for having the contents of the two SLOTS [d2] and [d3] combined into one sentence using ", *and*".)



Not only are the trigger conditions more complicated to test for, but, because the decisions are made "early" special records must be kept. These are the two attachments abbreviated *force-trace* and *force-vb-reduction*. We could alternatively do without these attachments by pruning *clause-10* there and then, leaving only its direct object. However, to do this would be effectively to erase all evidence that the [subject] and [verb] had ever been present, and would make it impossible to trigger the reduction of following clauses that had a parallel structure.

2.4 Coordinating alternate ellipsis strategies

There are often many alternative ways to reduce the same text: the examples below range from subject-merging through VPD and gapping to "one's pronominalization", and still leave out many possible variations.

"Both of us took a corner."
"Both you and I took a corner.[6]*"*
"I took a corner and so did you."

6. It seems to be an unwritten law of English—perhaps of English politeness—that whenever the nominal personal pronouns "*you*" and "*I*" are in conjunction they may appear in only that order. If we assume that this convention is not the product of continual conscious deliberation, then it must be integrated into the grammar so that it can apply without specific directives from the message. This we can do by establishing the appropriate test as part of [conjunct after-realization]: we look at the contents of two constituent conjunctions and if they are the pronouns in question (or their message-level precursors, which would require some initial collaboration with each domain as the grammar is assembled) and if they are in the wrong order we switch them. This routine is a "surface filter" in the sense of [permutter_thesis] [chomsky_lasnik].

"I took a corner and you did too."
"I took one corner and you __ another (one)."
"I took a corner and you took another one."

The problem is how these alternatives are to be controlled. Consider what would happen if we made them part of the "automatic" grammar—selected every time they were applicable: since their natural triggering sites are ordered top-down within the tree, the first to apply will always be subject-merging. But any time that subject-merging would apply, the others could have applied as well, except that by being first subject-merging will always have preempted them. Obviously, if the other constructions are ever to be used, the system of choices must be brought under some kind of coordinated control—there must be a common site for the decision where the alternatives can be deliberated over as a system.

I must admit to not having any clever ideas about how this coordination is to be brought about. The technique presently used in MUMBLE is assign each of the constructions a specific controller-variable. (Actually the assignment is to the grammatical-decisions that trigger them.) By including a test of this variable in the gating-conditions of each decision, it is possible to prohibit the selection of a construction on a per-context basis. A specific construction is selected by prohibiting the selection of all those that would preempt it.

As with most other choice systems in the grammar, the reasons one could have for selecting one kind of ellipsis over another are very difficult for people to articulate. One can point to a few differences: the first ("*both of us...*") is clearly marked because it drops all mention of the order in which the events took place; also the second to last ("*I took one corner and you another*") seems to emphasize the description "corner". But, at least for the moment, the others can only be treated as a synonym-set.

One would like to believe that when rhetorical goals and writing style are better understood it will be possible to eliminate this awkward coordination system in favor of specific goal-directed selection. That is, if we could determine what rhetorical effects are best served by each construction, and could label the context in the same vocabulary according to thes speaker's goals, then we would have very specific triggers where there is now only the most general trigger—"do it if you can".

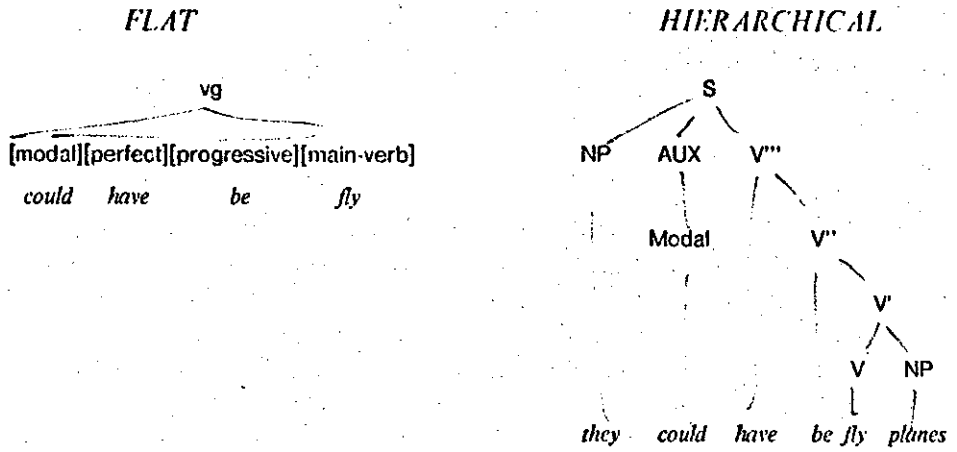
3. The Verb Group

The verb group has arguably the most complex set of realization constraints of any system in English grammar: its word order is ridgedly fixed; it has its own conventional intonation pattern [jon_&_??_aux_intonation]; it draws its elements from a uniquely wide range of semantic sources; and it has a set of filters that inhibit the simultanious realization of many combinations of those sources. To incorporate such a complex grammar into the linguistic component, you need a complex implementation—the implementation of the verb group is the most complex of all the grammatical phenomena handled by MUMBLE.

The term "verb group" is part of the theoretical vocabulary of systemic grammarians. In the text of an unmarked declarative clause, it corresponds to the region beginning at the end of the subject (plus any apposatives) and continuing until the beginning of the first object (or complement). All verb groups in this paragraph have been[7] underlined.

The primary difference between the systemic analysis and the average transformational one (there are a great many transformational analyses of the verbal system) is that it employs a flat constituent structure whereas the transformationalists employ a hierarchical one. This is illustrated below for the clause: "*They could have been flying planes*". The hierarchical example is adapted from [akmajian_li_10_1] pg.35]; notice that its "verb group" is not an independent constituent.

7. The linguistically sophisticated reader will have noticed that "*underlined*" was not included in the verb group even though it is a "passivized verb". This is because passives are being analyzed as adjective complements, capturing the similarity between, e.g. "*John was beaten*" and "*John was black and blue*" and reflecting it in a common constituent structure. This analysis says that the two texts are derived from comparable functional structures at the message-level. Transformationaly derived passives are created by the device of "predicate lowering" described later in this section.



There is considerable motivation for adopting the flat analysis when writing a grammar for language production. As discussed earlier (<constituent_structure_design_for_production>), constituent structure slots in the tree correspond one for one with msg-elmts in the message the tree is realizing. The number of slots that are available in each newly instantiated constituent should correspond to the number of subelements of the msg-elmts being realized. If a hierarchical constituent structure were going to be appropriate for the verb group, it would be because the message-level decisions that lead to a verb group unfolded in an equally hierarchical pattern. On the contrary however, in the present analysis the constituents of the verb group are assembled simultaniously, making a flat structure the most motivated.

3.1 Assembling the verb group

The basic problem of the verb group is that its constituents are decided upon at widely different times but cannot be brought together into their eventual surface structure format until the very last moment because the grammatical constraints that govern the format are defined only in terms of all constituents at once. This means that in the early stages, all decisions that affect the verb group, such as the selection of an adverb or the specification of tense or aspect, must be recorded on hooks attached to one of the nodes that will dominate the eventual verb group. When the controller finally reaches the [verb] slot, the records are scanned and reformulated as a constituent by a grammar-routine that is triggered as the slot is entered. Finally, once the new constituent has been incorported into the tree, the morphology routine applies a series of left to right adjustments as the words are produced.

In this section we will look at the mechanics of assembling a verb group, detailing the actions of the morphology routine on the output of [verb enter-slot]. I will only give one example here of accumulating specifications; the interesting cases will follow later.

Vocabulary

Theories of the human conceptualization of time and events abound in the literature[8] There is an agreement of sorts that speaker's point of view, the character of the time course of the event (completed, ongoing, instantanious, etc.), and the presence or absence of other temporal reference points are all involved, but there is no agreement about what primitives to use in characterizing these influences or what their relative importance is for the speaker with specific intentions in a specific pragmatic context. Certainly anyone who might adopt one of these theories for use an interactive program would be doing so experimentally.

In MUMBLE, I have taken a conservative stance and elected not to adopt any of the competing theories, but rather to couch the grammar in a purely syntactic vocabulary: a slight elaboration of Chomsky's phrase structure description of the auxiliary [Islt] pg.232].

VP_{aux} —> { past / count } <Modal> <have + en> <be + ing> <be + en>

Present dictionaries will mark a clause *have+ en* or *be+ ing*, rather than use a semantic vocabulary and mark it "habitual" or "limited" (terms taken from [quirk_&_greenbaum]). In any case, any "semantic" time relations would have to be translated into this syntactic vocabulary at some point during their realization, and thus grammar developed here will remain useful regardless of what "higher" theory is eventually adopted.[9]

The time-specifications presently accepted by MUMBLE are listed below in the order in which they appear in the *full_verb-group* constituent-schema.

---

8. I personally have been influenced by at least [kahn_time][3pt_time_ref][fillmore_time_lecture][pink_time_book] [wosenschagel_time].

9. Specifications of time relations that involve the speaker's point of view and her or his goals would naturally be a part of the message. The incorporation of one of these theories into the grammar would effectively constitute an elaboration of the interlingua (<the_beginnings_of_an_interlingua>) and would involve the presumption that all of the speakers that used the linguistic component would be comfortable with that conceptualization.

SLOT-NAME	*LEGITIMATE CONTENTS*
[pre-aux-adv]	an adverb, an adverbial phrase
[modal]	*will, would, can, could, should, etc.*
[have + en]	*have*
[be + ing]	*be*
[be + en]	*be*
[mvb]	a verb
[post-mvb-adv]	an adverb, an adverbial phrase

DYNAMICALLY POSITIONED SPECS.

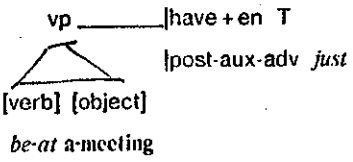
HOOK: not	*not*
HOOK: post-aux-adv	an adverb

One case of a "higher-level" time specification does appear in MUMBLE, namely the instruction to "emphasize polarity" (pg.<attachments_and_default_decisions>). There the entry making the decision that the positiveness (or negativeness) of the text should be specially marked is not itself in a position to implement the marking in its ultimate vocabulary (i.e. stress-a-word, add-"do"-to-vg, inhibit-contraction) because the locations in the tree where the marking would have to be made do not yet exist. What the entry does instead is to add a hook, *emphasize-polarity*, at closest location that does exist, expecting a grammar-routine(s) from the general purpose grammar to later notice the hook and continue the marking process.

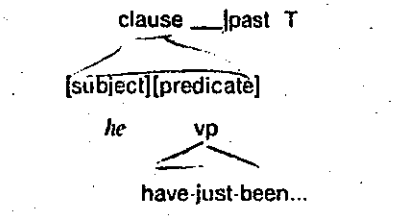
Strictly speaking, the ultimate vocabulary of emphasizing polarity is known only to the morphology routine in the sense that it is the only part of the linguistic component that is able to stress a word by capitalizing it or to add "*do*" to the verb group. Thus the interpretation of the high-level specification "emphasize-polarity" would be done by the morphology routine as a variation within its normal task of assembling and printing words associated with the first word of the verb group. Other high-level specifications would be handled in an analogous manner: first introduced into the tree as an uninterpreted symbol (a hook), then noticed and translated into the purely syntactic vocabulary by a grammar-routine acting at a point where that vocabulary makes sense. The fact that emphasize-polarity cannot be realized except by the morphology routine (acting at the last minute as it were) is probably unusual; a concept like the "narrative past" or "habitual" could be translated into a syntactic vocabulary at a much earlier point, e.g. at the clause level, and would then use the normal routines for the rest of its implementation.

Sources for the specifications

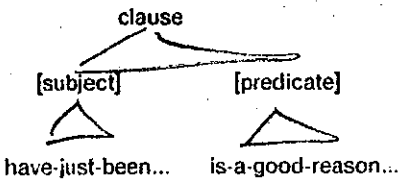
Basically speaking, there are two kinds of message-level sources for verb group specifications: the event or action that is the basis of the [predicate], and modifications to that event (specializations, negations, modalizations, qualifications, temporal relativizations). Events (or more typically, abstract predicates) can specify more than just what verb will be used: In a domain like the Personal Assistant, predicates can naturally include an aspect specification as in to-have-just-been-at-a-meeting, which is intrinsically perfect ("*have+en*") and modified by an adverb. (This predicate would be quite useful: from it the Assistant could directly infer that people to whom it applied would be (1) tired, (2) either exilarated or frustrated but in any case not emotionally normal, and (3) unlikely to be interested in going directly to another meeting.) Below is a snapshot of what the phrase for that predicate would look like if it were instantiated in issolation.



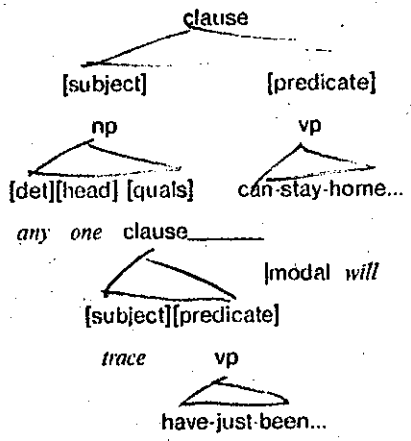
Modifications to this predicate can come from many sources. If it is used to describe an event, the time of the event relative to the time of speech (somehow encoded in the message) would add tense or modal specifications above the verb phrase which the [verb enter-slot] routine to take into account. Specifications could also come from linguistic context, as when the verb phrase is the context of an [adjunct] or a [complement].



"He had just been at a meeting."



"Having just been at a meeting is a good reason not to want to go to another one."



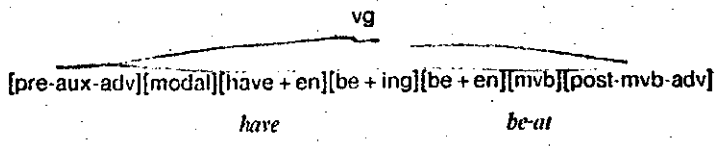
"Anyone who will have just been at a meeting can stay home the next day."

From hooks to constituent structure

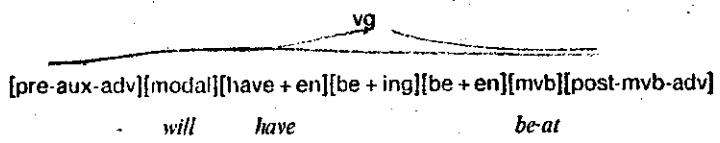
Once the controller has reached the [verb] slot, it is no longer possible to add specifications to the verb group: no further msg-elmts will be realized before the controller has passed through the verb group and it has been printed into the text. At this point, a grammar-routine is triggered whose function is to accumulate the specifications recorded on the hooks, apply the grammatical constraints, and construct a verb group constituent accordingly.

Given that the responsibility for producing the ultimate text for a verb group is divided between this grammar-routine and the morphology routine, only one kind of grammatical constraint needs to be fixed at this point, and that is constituent order. Order, as always, is implemented through the use of a constituent-schema that embodies the correct order of constituents. In the case of verb groups, order is never contingent on the presence or absense of constituents and thus only one maximal constituent-schema need be used (assuming we are not

disquieted by empty slots). The schema was given earlier in the discussion of the proper vocabulary. Of the last three examples, the first two lead to the same pattern:



and the third is only slightly different:



In performing its "gathering" function, the [verb enter-slot] routine must be sensitive to the grammatical context in which the verb group appears. Not all contexts will support all possible elements of the verb group: infinitive contexts, for example, may not have modals. Whether, in the long run, one will actually have to actively check for these conditions is an interesting question: is there a consistent semantic or pragmatic quality about what is syntactically an "infinitive context" that may already be part of a speaker's message vocabulary? If that is the case, then the "no modals" constraint would already be implicit at the message level and not need to be stated within the grammar of the linguistic component. If, on the other hand, infinitives were an artifact of the English language *qua* language, then the constraint would be required and the question of whether its application thwarted the speaker's intentions, perhaps requiring some kind of deliberation over alternatives, would become important.

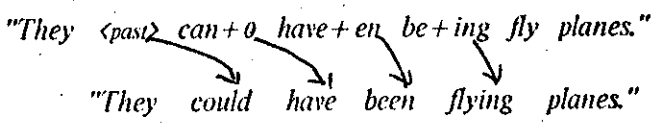
3.2 The role of the Morphology Routine

The morphology routine is responsible for two aspects of the implementation of the verb group: (1) verbal elements whose position is defined relative to "the first word of the verb group, e.g. tense, negation, and some adverbs; and (2) "aux-hopping", arranging that the aspectual and passive suffixes ("*+ en*" and "*+ ing*") are attached to the appropriate verbs. The rules that govern both of these phenomena are encoded directly into the morphology routine rather than involving a declarative representation and an interpreter as is the case with most syntactic facts; this is in keeping with the policy that no other parts of the linguistic component are expected to need to reason about these aspects of morphology.

Aux-hopping

The morphology routine is configured as a finite state transducer: it receives words one at a time as they are passed to it by the controller, buffers them (to apply contractions and ultimately to stage allophonic variations), and them passes them to the output stream for printing. Whenever it receives a word it may notice either to the identity of the word or to some fact about the linguistic context (e.g. the identity of the current-slot) and then react by changing its state so that it will behave specially when receiving the next word. This property of the routine makes it a natural place to implement aux-hopping.

The phenomena of aux-hopping derives its name from the transformation proposed by Chomsky as part of his original analysis by which the suffixes of the "words" positioned by his phrase structure rule, e.g. *have+en*, *be+ing*, and *be+en*. (The morpheme "*en*" represents the past participle form and will be realized as "*ed*" or "*en*" depending on the specific word they are adjoined to.) The transformation took each suffix in turn starting at the left and "hopped" it over the next word and adjoined to it.



To translate this transformation into the form of a finite state transducer, we note the value of the current-slot of each word of the verb group in turn and use it to specify what suffix to adjoin to the next word that is received. Leaving aside the case of the first word of the verb group for the moment, let us look at the example: "*could have been flying*".

Event #	*Current State*	*Word Received*	*Word Printed*	*New State*
1.	⟨first-word-of-vg⟩	*can*	*could*	null-suffix
2.	null-suffix	*have*	*have*	en-suffix
3.	en-suffix	*be*	*been*	ing-suffix
4.	ing-suffix	*fly*	*flying*	null-suffix

The morphology routine is the only part of the linguistic component that has a text-level description of context rather than syntactic, constituent structure-level description—it notices only the sequence of lexical leaves of the tree and none of the complex relations that led to their selection. This means that linguistic phenomena that depend only textual sequence can be advantagously stated at this level.

Dynamically positioned elements

In addition to receiving words from the controller, the morphology routine receives notice of the passage of the controller through syntactic regions, e.g. it is told when the controller begins to traverse the constituents of the verb group.[10] Given this information, the routine can trivially define the grammatically important concept "*first word of the verb group*", namely that it is the next word the routine receives after it has been notified that the verb group has been entered.

When this word is recieved, a special set of processes are activated that scan the tree (via controller-variables) looking for hooks indicating the specification of past tense, negation, or a post-aux adverb. The past tense is used to arrive at the correct print name for the "first word of the verb group" (provided it is not overruled by the linguistic context; see below). The adverb, when present, is printed directly by the morphology routine after it prints the word it received and before receiving the next.

The grammar rules for negation are more complex. When there is an auxiliary present, the negative morpheme follows it as "*not*" or may be contracted to it; however, when there is only one word in the verb group—the main verb— the rule changes and the word "*do*" artificially introduced as the first word of the group which the negative then follows. In MUMBLE's morphology routine, these rules are represented procedurally and are entwined with the rules for contraction and polarity which also refer to negation. The complexity of the contingencies involved make this one place in the grammar where a well conceived procedure is more perspicuous to the human designer than any declarative listing of the rules.

3.3 Computing 'tense'

The printed (or spoken) form of the first word of the verb group varies according to the grammatical context of the verb phrase containing it. In MUMBLE's grammar, all that has been necessary to define this context has been a controller-variable pointing to the slot that contained the verb phrase; the slot's features indicated the kind "tensing" that was to be performed.

As elsewhere, the grammar writer's selection of features involves a tradeoff that is partly a matter of physical labor and partly a matter of representing generalizations. On the one hand we could identify the specific locations where embedded verb phrases (and therefore embedded verbs) could occur, and incorporate a list of those slot-names into the morphology routine. Anytime that we later added a new slot-name to the grammar we would have to ask whether a slot so labeled would ever contain a verb phrase and if so what form its verb should have. Should it happen than the same slot-name (appearing presumably in different constituent-schemas) can

10. In MUMBLE, this notice is implemented by a controller-variable that is set as part of [verb enter-slot] and read by the morphology routine when it receives the next word. In a machine implementation that permitted the morphology routine to function in parallel with the controller (a design that would be appropriate for speech production where the amount of work the routine would have to do would be much higher) the notice might be given by message passing just as though the information was a word itself.

contain verb phrases with more than one kind of tensing then this "literal list" technique would no longer be sustainable and we would be forced to move to the alternative technique where a fixed set of slot-features is created—one for each different kind of tensing required morphologically—and then added to the features properties of the slot-names actually used to contain the verb phrases should they be different. Here, each time a new verb phrase containing slot-name is added we must ask which of our fixed set of features corresponds to the kind of tensing it requires and then add that slot-feature to it. If it can use multiple kinds of tensing, we do not add any of the features permanently but instead arrange that the contextual contingencies which dictate the choice be associated with a grammar-routine that will add the appropriate feature once we know what it should be.

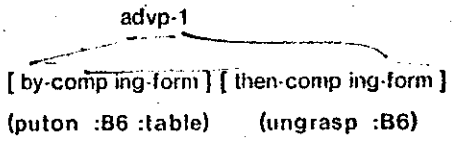
In the course of developing MUMBLE there has been a shift from using specific slot-names to using a fixed set of syntactically motivated slot-features. A concession to utility (adding the features is a fair amount of work) has been the adoption of a marked-unmarked convention whereby a feature only has to be added when a form is required other than the "unmarked" declarative form that allows modals and past. I make no claims for the optimality of this set of features.

MUMBLE presently uses the following features to control the form of the so-called "tense-carrier", the first word of the verb group (pre-aux-adverbs discounted of course). Note that this is actually a feature "system" in that some of the features permits the further choices: *participle* requires a choice between "*ing*" and "*en*", taking the later when the embedded verb phrase has the hook *be+en* indicating passive; the unmarked case shows the past tense if it is present, otherwise is shows person and number agreement with the current-subject.

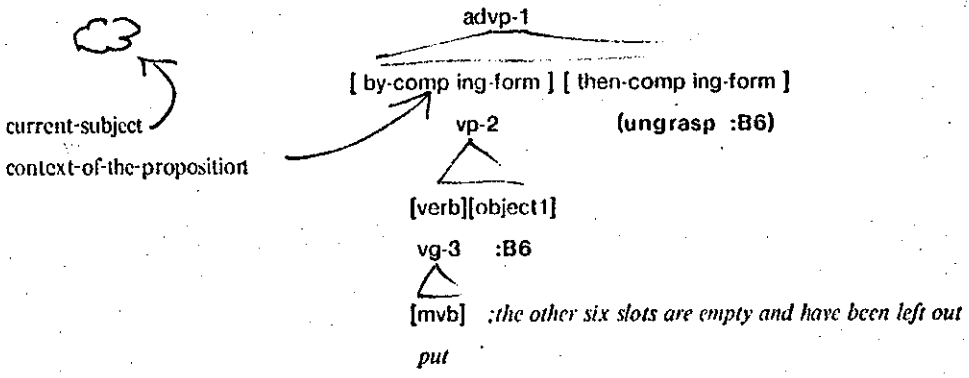
<no feature>	{ past, subject-number }	
participle	{ *+ing, +en* }	
infinitive	<unmarked form>	*;the "to" is supplied by a grammar-routine*
ing-form	*+ing*	
head	*+ing*	*;as in "head of a noun phrase". Gerunds and the complements of verbs such as "like" are analyzed as verb phrases that have undergone a transformation into noun phrases.*

To see briefly how this system works, let us look at part of an example that we will return to later (pg.<isomorphic_message_elements_n.y.w.>); it is the output of a very simple entry that answered a "how" question by specializing a neutrally represented list of expressions from an internal action history by putting them in a linguistic context (a canned phrase) that would correctly adjust their tense and aspect. Because it is an answer to a question, the body of the

question (i.e. "*how did you do it?*") can be assumed in the answering text and all that need be given its the new information: "*(I did it) by putting a large red block on the table then letting go of it.*". The canned phrase used is an adverbial phrase beginning with one constituent "bound" by the function word "*by*" and continuing with an arbitrary number of constituents bound by "*then*" (the definition of arbitrary length constituent-schemas is on page <schema_with_arbitrary_numbers_of_constituents_n.y.w.>). The snapshot below shows this phrase just after it has been instantiated with two msg-elmts representing actions from the history. Notice how the labeling of the slots does the double duty of (1) arranging for the printout of the function words, and (2) indicating the proper form of the verb.



In the next snapshot, the first msg-elmt has been realized, the controller has passed through [verb enter-slot], and we see the current context as viewed by the morphology routine.



"By... //"

The morphology routine knows that "*put*" is the first word of the verb group, consequently it applies its its "tense-computing" procedure, using the controller-variable context-of-the-proposition to access the slot that contains the verb phrase and from that slot the features that control tense. In this case the relevent feature is *ing-form*; it initiates a morphographemic process that adjoins the suffix "i n g" to the pname of the verb. (In MUMBLE, verb pnames are stored in their infinitive form.) A common process is used for the adjunction, and it is there that morphographemic combination rules such as doubling final consonants after stressed vowels are stored.

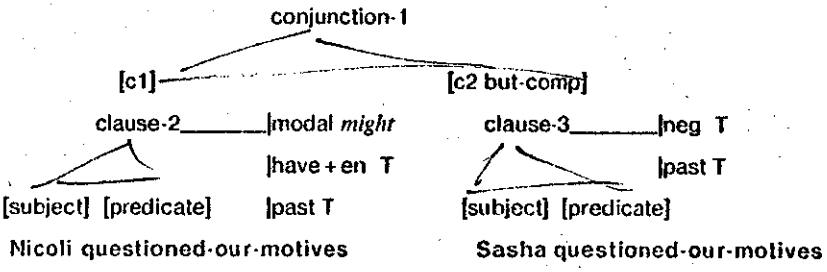
Reduced verb phrases

Operations such as ellipsis or equi-np-deletion create situations where only a fragment of the normal verb group specification is present. For example we can say:

"Nicoli might have questioned our motives, but Sasha didn't"
"I kissed everyone you told me I should."
"You can't tell them who ___ not to sell their oil to."

Except for the last case involving negation and equi (whose implementation is admittedly baroque; see pg.<equi_neg>), these cases of reduced verb phrases can be implemented within the present technique with only one addition, namely modifying the morphology routine to treat an empty verb group as though it were a virtual main verb, i.e. having it draw the auxiliary verb "*do*" as a carrier for tense, count, or negation. Consider the first example:

Initial Tree



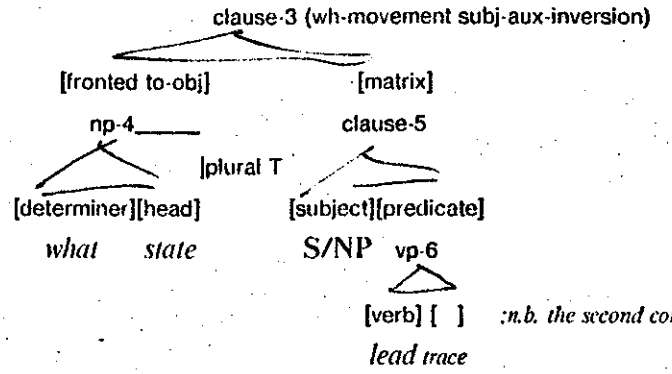
Verb phrase deletion will be triggered when the controller reaches the second conjunct and notices (in the subsequent reference routine, pg.<pronominalizing_predicates_n.y.w.>) that questioned-our-motives has appeared before and could be safely omitted. The subsequent reference routine implements this ellipsis by returning a trivial verb phrase that consists of only one constituent, a [verb] with null contents. When the morphology routine sees the combination of an empty verb group and a unmarked context specifying negation and past tense, it introduces the word "*do*" and finally produces the pname "*doesn't*".

3.4 subject-verb inversion

In indo-european languages, questions can be formed textually by exchanging the position of the subject of a clause and its verb. In modern English, only the "first word of the verb group" is exchanged rather than the whole group, and if there is only a single main verb in the verb, the auxiliary "*do*" is exchanged and the main verb left in its place. Of course, as the reader must by now have anticipated, in this theory there cannot be a literal "exchanging" operation but rather some annotation special to questions must arrange that this "preposed auxiliary" and the subject appear in their ultimate positions from the start.

The realization of the preposed auxiliary is done by a grammar-routine triggered at [subject

enter-slot].[11] Below is a further snapshot from the sequence used to illustrate wh-questions (pg.121). It is taken just at the moment when the [subject] is entered.



"To what state... // "

The "preposed" (also called "displaced") auxiliary is a classic example of a "discontinuous constituent": it consists of two segments, separated in the text by the interveening subject noun phrase. Discontinuous constituents have always been a nuisance for grammar writers and the present case is no exception: what was a tidy algorithm for the contiguous case now becomes awkward and ad-hoc. The problem revolves around access to information: In the earlier contiguous case, we did not begin to assemble the verb group until all of its specifications had been completed and the lexical form of the main verb identified; in the discontinuous case, we are forced to "lookahead" for facts like subject number, intrinsic modality or aspect of the predicate, or the mode of the main verb (i.e. is it pronominalized with "*be*" or "*do*"?).

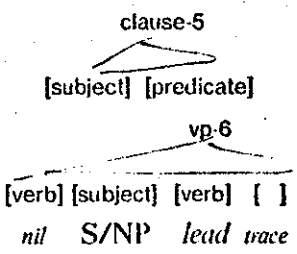
If the needed information were available directly from the message-level representations of the subject and predicate, then the stipulation of incremental realization would not be "damaged" since no realizations would have to be performed ahead of their natural order as established by the structure of the tree. (Recall that the [subject] and [predicate] constituents are readily available at the point in the tree where the information would be needed.) This would mean, however, that concepts like lexical number,[12] modality, aspect, and state versus action ("*be*" vs. "*do*") would have to be either part of the interlingua or readlily computed from the relevent entrys in the dictionary—something that does not seem unreasonable, and which is already true to a certain extent in the Macbeth domain and the KL-ONE domain. If the information is not

11. Depending on the grammar writer's proclivities toward permiting contingencies within grammar-routines as opposed to having slot-features added to lower slots by higher nodes or slots, the routine for creating the preposed auxiliary may either be a standard part of the actions associated with a "subject" or it may be associated with a special feature that "migrates" down form *wh-movement*. I have elected the former.

12. A critical case would be the determination of whether a choice such as the one between saying "*these bananas are ripe*" versus "*this bunch of bananas is ripe*" is made by the speaker and encoded in the message or made in the dictionary and not made until the subject is actually realized. My intuitions suggest that it is made by the speaker.

available, then there is no choice but to perform the lookahead, perhaps managing to stop with the verb group and not continuing on into the rest of the verb phrase. This would not be irreparable damage to the stipulation since the left to right dependencies between the subject and the verb group and the subject are minimal and it might be argued that they could be realized in parallel. It would not, however, be easy to perform this lookahead operation with the present control structure.

Technically, the preposed auxiliary would be constructed by having the initiating grammar-routine on [subject enter-slot] create an expeditious but grammatically invisible[13] node that "lowered" the contents of the given [subject] into a "new" [subject] slot which was the second of the two constituents of the new node, the first being [verb].



"To what state... // "

By using the same slot-name for the preposed auxiliary as we use for the "normal" verb group, we will activate the same grammar-routines and send the same initializing signal to the morphology routine. The only difference is that now we are presuming that the operations used to access hooks like *past* or *have+en* are able to extract their information from elmt-instances as well as nodes. In order to complete this technique, one further item of state information must be provided to the morphology routine, namely a "switch" to insure that the "first word of the verb group" is realized only once, i.e. it must not be repeated once the controller has moved inside the [predicate] and the rest of the verb group is resumed.

Now that this considerable amount of machinery has been developed to deal with subject-verb inversion, it can be applied to other phenomena. The remaining parts of this section will each illustrate a different case. Linguists might feel that this technique of submerging constituents and having state information in the morphology routine is unwarrantedly baroque, but I would argue to the contrary that it is highly motivated by the special circumstances of language production (at least as I understand them) and is no more baroque than many of the intermediate constituent structures that linguists have themselves proposed over the years.

13. It would have no features and thus no grammar-routines.

Subject-verb inversion occurs in other constructions besides questions, all of which appear to be some kind of topicalization:

"Never have I been so humiliated."
"...and so did you."
"On the left is a second bathroom."
"Half-hidden by the mist was a tall, one-eyed man with a raven on his shoulder."

These constructions can be implemented by the same device as questions are, namely having their transformation add the feature *subj-aux-inversion* to the clause.

3.5 Tag questions

A "tag" question consists of a declarative clause followed by a phrase made up of the "first verb" of the clause, now with opposite polarity, followed by a pronominal copy of the clause's subject.

"You are going to pay me back, aren't you?"
"You won't forget about our appointment, will you?"

Psychologically, tag questions are particularly interesting because with certain intonations they are very probably the result of a last-minute decision by the speaker to change their hesitant statement into a definate question.

Given the apparatus just described, a tag phrase can be constructed by adjoining to the end of the clause a node with the feature *subj-aux-inversion* and the hook *neg*; the node will have one constituent, a copy of the current-subject specially marked *pronominalize* (pg.164). Provided the grammar-variables of the clause are still accessible (pg.<grammatical_relations_over_variables_rather_than_tree_structure_n.y.w.>), this new node will be expanded just as if it were the regular subject of a question and the correct text for the tag will be produced.

3.6 Existential *there*

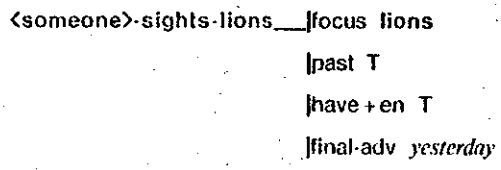
The facts Existential "*there*" has, like passive or indirect object movement, been a part of the central syntactic analyses of every significant school of linguistics (at least since the late 1950's). The concensus appears to be that semantically the construction derives from a for some reason unrealizable intransitive predication of "*to be*" or "*to exist*", as in:

exists("a mouse in the house") => *"There's a mouse in the house."*

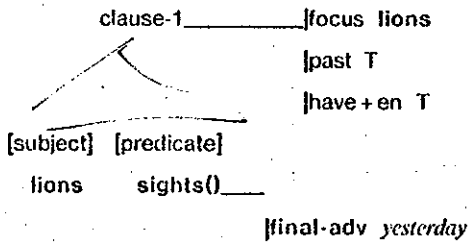
There is not, however, any concensus or even much discussion of whether "*there*" is acting as an operator, i.e. its argument is a natural unit that could be realized on its own, or is acting as a predicate, in which case we would not expect its argument(s) make sense as a unit. I will take it to be an operator.

Syntactically, the word "*there*" functions as the subject of its clause, e.g. it forms tags: "*there's a mouse in the house, isn't there?*", while at the same time, the verb that follows "*there*" (invariably an auxiliary rather than a main verb) agrees in number with the "direct object" (if that is its correct description), e.g. "*There are dozens of mice in this house.*". This "direct object" is invariably indefinate; though it is not at all clear whether this is due to a syntactic constraint or to a uniform policy by the speaker to only apply the operator to descriptions of generic objects. I will take it to be the speaker's policy.

The analysis Having taken existential "*there*" to be an operator, it is natural to implement it as a transformation, thereby making it largely independent of the realization of the object description that is being declared to exist. As an example, imagine that the speaker started with an observation that might be realized as "*Someone had sighted some lions yesterday*",[14] and recast it to focus on the lions: "*Some lions were sighted yesterday*". This would initially appear in the tree as shown below (assuming, of course, that it was to appear as a toplevel sentence).



which becomes:

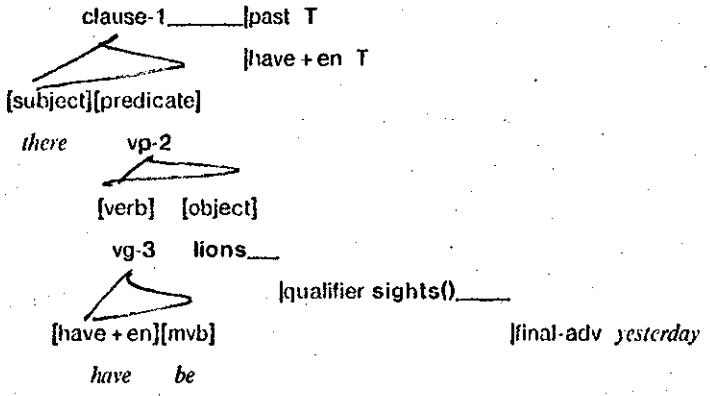


Now imagine that the speaker wants to apply the "existential-there" operator to that initial elmt-instance. Technically we can say that this is done by adding yet another hook, however the precise technique used doesn't matter so long as the decision-rules of the "there-insertion transformation" are triggered when that elmt-instance is realized. The transformation will apply after the "basic" realization of the element has been selected, i.e. the clause shown above, and will function by editing the schematic description of that clause so that when the entry's choice is finally instantiated and put in the tree, the transformed version will be present from the start.

---

14. Contrastive stress on "*had*".

The editing operation consists of the creation of a new clause node with a [subject] slot and a [predicate] slot, and filling the [predicate] with a verb phrase node with a [verb] slot whose contents are the word "*be*", and a (non-passivizable) [object] slot. The [subject] is filled with the trace "*there*",[15] and the trace linked to the contents of the original [subject]. This device will implement the necessary number agreement. Any temporal hooks on the original clause are transferred to the new clause; crutially however, any hooks that might be visible on the contents of the [predicate] remain where they are. The transferred hooks will now be realized as part of the verb group of the new clause. The original [subject] and [predicate] constituents are very throughly rearranged: the contents of the original [subject] slot are mapped into the new [object] slot, and the contents of the original [predicate] are mapped into a hook on the original subject elmt-instance marked *qualifier* whose effect will be to make the original predicate a relative clause of the subject.



"There had been lions sighted yesterday."

3.7 Adverbs

The subject of adverbs and their placement in a text is much more extensive than I have been able to analyse or implement in MUMBLE. A simple facility was installed whereby an adverbial modifier to an entire clause could be positioned at either beginning, post-auxiliary, pre- or post-verb, or final positions as decided by synonym-set. A hedge like "*possibly*" would be realized in this way, the decision presumably coming as part of a shopping-list for the proposition being hedged and applying after the matrix of the clause had been created. However it is quite clear that the positioning decision for an adverbial phrase depends upon subtle but crutial differences in scope, intention, and discourse context. Differences that I am only dimly conscious

15. The analysis of existential "*there*" as a trace appears in some of the transformational literature, e.g. Dresher and Hornstein [dresher_&_hornstein_li_10_1] pg.67] analyze it as a rightward movement (w.r.t. a string) of the original subject NP over its verb, which leaves a trace that is then covered by the lexical item "*there*".

of as a writer even though I make such decisions all of the time.

Early on in this research however, I did notice one phenomenon that does not seem to have been noticed elsewhere in the lingusitic literature: this is a commonality between post-auxiliary adverbs and the so-called "raising verbs" that I have analyzed as involving a grammatically triggered adjustment transformation I call *predicate lowering.*

Consider this set of sentences, all of which involve the modification of a common predicate along a dimension we could call "extent of progress made".

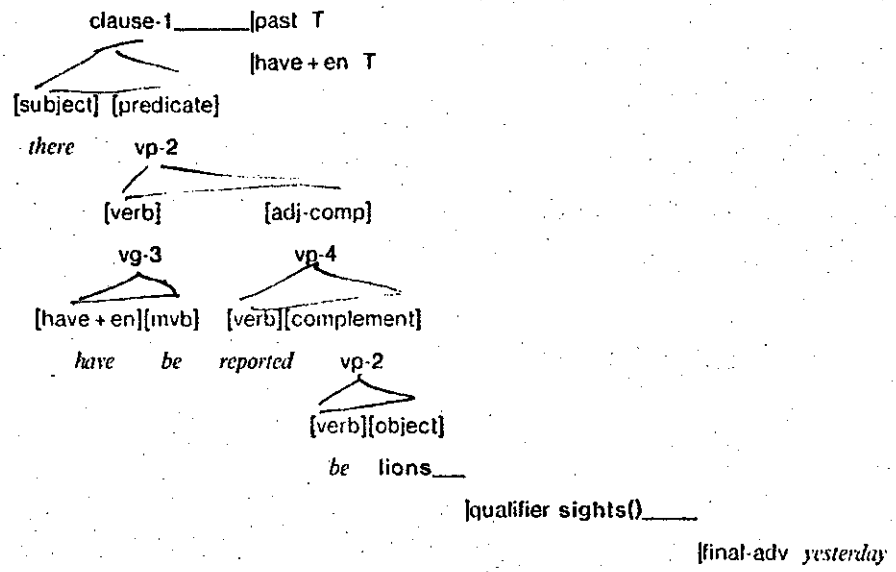
"I'm starting to write my thesis."
"I'm rapidly writing my thesis."
"I'm in the midst of writing my thesis."
"I seem to be writing my thesis 24 hours a day!"

Only one of these modifiers, "*rapidly*", would be normally be called an adverb, yet all four (and dozens upon dozens like them) are performing an adverbial function. If their functions are the same, is it not reasonable to expect that their origins in the message will be the same as well? Would it be reasonable for a speaker to want to derive "raising verbs" like "*seems*" or "*be reported*" as though they were modifiers like "*rapidly*" rather than verbs like "*write*"? I would say yes, though some people would disagree.

Even if (some) raising verbs are treated functionally as adverbs, it remains true that syntactically they appear in the text as true verbs: they are affected morphologically by the same contexts, and they participate with other verbs in aux-hopping (e.g. "*If I have seemed unusually tired today, it's because I was writing my thesis all night.*"). However, as elsewhere in this linguistics component, this disparity between message-level function and surface-level form can be reconciled by the judicious use of procedures attached to the functional vocabulary that are empowered to adjust the tree to bring about the correct form.

The procedure for predicate lowering is in this case attached to post-aux-adv, which is effectively an event within the morphology routine comparable with enter-slot or after-realization in the controller. This routine looks at the properties of the "adverb", say it is "*be reported*", and determines whether it has a complement-type property; "*report*" will, and its decision will list one default transformation, namely marking the [complement] *infinitive* (see pg.<message_level_equi_n.y.w.>). At this point, the morphology routine applies predicate lowering: The details of what the transformation does are quite baroque and I will not burden the reader with them here; those interested will find them in the appendix. Suffice it to say that once the dust has settled the constituent structure has been modified to be as we see it below. "*Be*

reported" is applied to the earlier sentence about the lions.[16]



"There had been reported to be... //"

This sort of adjustment transformation can be used elsewhere as well, for example in the placement of an "adverb" like "*I think that ()*", or in sentence level adjunction. Futher examples are developed in the appendix (pg.<adjustment_transformations>).

16. If one insists that "*report*" derives from a predication of the same stature as (and hierarchically superior to) <someone>sights-lions in the message, with the exists operator applying to the whole gimish, then given the stipulations of this process, the message-level relatation report must have multiple arguments– it must pick out at the message-level what is being reported on and what is being reported about it. Given this explicit breakdown, the normal "there-insertion" transformation would produce the desired constituent structure; however, lions and sights() would now be independent and a different analysis would be needed to construct the noun phrase-qualifier phrase constituent structure, or more to the point, to also be able to construct the transformational alternative with extraposition: "*It was reported that lions had been sighted yesterday.*".

4. Pronominal subsequent reference

Whenever the speaker wants to refer to some entity or proposition, it must find a phrase that will both provide an adequate description and be appropriate to the context. This process of producing *references*[17] will vary according to whether the reference is initial and subsequent. *Initial references* introduce new entities into the discourse, while *subsequent references* are later instances of already introduced entities. An initial reference can take almost any form so long as the audience will be able to recognize it. Choosing this form, however, can be very difficult: the speaker must be certain of the "mutual knowledge" [best_mutual_knowledge_ref] it shares with the audience in order to make a judicious choice of which properties to use in the description. How this is done is very poorly understood. When talking or writing about unfamiliar things or to unfamiliar audiences, it is even difficult for people. Subsequent references are another matter. They are very highly grammaticised: involving specialized forms and restrictive conventions, and their use is automatic and remarkably consistant. Because of these properties, techniques for making subsequent references are a logical candidate for incorporation into the linguistic component as another "automatic" facility that dictionary designers can take for granted.

4.1 Coordination with the realization procedure

As charted on page <flowchart_of_the_realization_procedure_n.y.w.>, the realization procedure divides initially into two separate paths depending on whether on not the message element being realized has ever been mentioned before. If it has never been mentioned, control is directed to the "main stream" where the element's entire ENTRY is interpreted following the procedure described in section II.6.2.vii. (N.b. ENTRYs should be written with the expectation of producing "initial" references.) If it has been mentioned, then one of three things happens: (a) the heuristics of the pronominalization routine decide that a pronoun can be used and do so; (b) the heuristics decide against a pronoun but there is some other subsequent reference strategy that can be used (this usually entails rejoining the main stream after selectively modifying some DECISIONS); or (c) there is no subsequent reference strategy available and the regular main stream realization is used as a default. This section will describe how the decision to use or not use a pronoun is made; alternate subsequent reference strategies (e.g. definite determiners or adjectives like *such* or *another*) are the subject of section non_pronominal_subsequent_reference_n.y.w. in the next chapter.

The decision to think about making a decision — The heuristics that govern whether a pronoun is used are evaluated in two stages according to how difficult the PREDICATES are to compute. The

17. Researchers in speech-act theory refer to the making of "reference acts"; see for example [brian_toronto][cohen_thesis].

first stage is based on a set of very simple tests against immediately available data. Very often the pronominalization decision can be determined from these tests alone, and the not-inconsiderable overhead of the second-stage heuristics can be avoided.

The tests fall into four categories:

(1) Is the ELMT-INSTANCE specifically marked (via an attachment) either to be pronominalized or not? Because of special rhetorical circumstances, the pronominalization decision is occasionally made as part of a much earlier CHOICE. Recording that decision for implementation now that the instance has been reached avoids redundant recomputations.

(2) Is the message element this is an instance of ontologically of a sort that cannot be pronominalized? For purposes of reference, message elements can be divided into "references", "descriptions", or "other" according to their role in the expert program's model of the world. Of these, only references denote something of the sort that a pronoun can be used for. Descriptions have their own sort of "pronominalization" (e.g. *one* and *such*) and this test will dispatch to a routine for them (pg.230). Under "other" come those message elements that act directly as instructions to the linguistic component or to a specific entry and some that acted as "aliases" for other message elements. None of these are ever realized initially as phrases and thus never subsequently as pronouns. This kind of ontological information is usually not explicit in the element's structure and is determined by an interface function: *elmt-reference-type*, which can use speaker or expert specific criteria to determine which is the case.

(3) Can we tell from the SLOT in which the ELMT-INSTANCE appears that a pronoun cannot be used grammatically? Pronouns are a kind of noun, and consequently can appear only where it would be grammatical for a noun or noun phrase to appear. This property is indicated in MUMBLE by the SLOT-FEATURE *nominal*—any SLOT without this feature cannot take a pronoun. Some of these SLOTs will, however, be plausible sites for verb phrase deletion or other sorts of pro-forms and the routines for making those decisions are dispatched to when they are the *current-slot*.

(4) Finally, is this a subsequent reference to the message element—does it have a RECORD that can be accessed via *elmt-discourse-history*? If this is true and none of the previous tests apply, then we move on to the second stage for more elaborate tests. Otherwise, this ELMT-INSTANCE is an inital reference and the main stream realization process is used.

The reader may have noticed that all of these tests are for anaphoric reference, i.e. the full noun phrase always preceeds the pronoun. In restricted syntactic circumstances, English does permit the optional use of cataphoric reference (so-called "backwards pronominalization"), as in:

"In his house, John smokes pot."

"Because Lady Macbeth persuaded him to do it, Macbeth murdered Duncan."

However, this usage is so restricted and so rhetorically motivated, that it is not used in MUMBLE unless explicitly planned for via attachments to the affected element.

4.2 Describing anaphoric relations

The decision of whether or not to use a pronoun is based on an analysis of the relationship of the present instance of the message element to its previous instances and a consideration of any nearby potentially distracting references to other entities. The current context as described by the CONTROLLER-VARIABLES is compared with the properties recorded on the element's discourse-history, and then evaluated according to the weighted votes of a body of heuristics such as: "not most recent", or "was pronominalized last time".

The first step in this analysis is the derivation of a set of *abstract features* from the raw data of the CONTROLLER-VARIABLES and RECORDS. This intermediate step greatly simplifies the statement of the heuristics by abstracting away irrelevant details and separating the data-access routines from the actual heuristics.[18] For example, it is all the same to the present heuristics whether the previous instance was actually realized as a noun phrase, a trace, or a nominalized clause. Consequently the feature-analysis merges all three cases into one feature: *was-a-thing* (which the heuristics distinguish from *was-a-proposition*).

Other features that MUMBLE computes include:

- **Measures of recency:** The antecedent is either in the *same-clause*, *same-sentence*, *same-paragraph*, or "*stale*" relative to the *current-instance*. (*Stale* is arbitrarily defined as in the same paragraph but earlier than the previous sentence. It is a crude but plausible meaning-independent measure of when a message element will have to be reintroduced.)
- **Intra-sentential syntactic relations:** the antecedent either *commands* the *current-instance* (i.e. it is a constituent of (one of) the node(s) that dominates the current position of the controller) or it is *relatively-subordinate* (every other case, e.g. the earlier instance is part of an introductory adjunct or a sentential subject). English grammar dictates[19] that when *comands* holds a pronoun must always be used, even if an ambiguity might result, e.g. the clause: "*Mortimer persuaded Ferdinand that he was the best man for the job*" is grammatical even though it is ambiguous out of context, while "*Mortimer persuaded Ferdinand that Mortimer was the best man for the job*" is unacceptable unless there are two people named *Mortimer*.
- **How the antecedent was realized:** *was-a-thing*, *was-a-proposition*, or *was-a-pronoun*.
- **Relative position in coordinated regions:** when the *current-instance* occupies a parallel position to its antecedent in the *previous-conjunct*, the identification between the two is typically strengthened. "Parallelism" is defined in terms of occupying SLOTs with the same slot-name at identical depths within adjacent clauses of the conjunction. Of course, when the antecedant was part of a conjunction and the instance is not, pronominalization is ruled out.
- **Discourse status:** all subsequent references to the current intended discourse focus are

---

18. It also makes it unnecessary to rewrite the heuristics every time there is a change to the design of the records or a new definition of a feature, which is of enormous practical benefit.
19. This is the familiar "precedes and commands" rule of Langacker [langacker_p_and_c]. Recall that because of the way that the controller traverses the tree earlier instances will necessarily precede the current instance in the left to right sequence of the text.

pronominalized as part of the definition of focus. An unfocussed antecedent that occurs in a constituent positions that is one of the preferred points for the introduction of a new focus: subject, direct object (theme), or syntactically marked positions like the object of existential *there*, are marked as a *potential-focus*.

In multi-sentence discourses, a given message element may accumulate several "non-stale" RECORDs in its discourse history, at which point it becomes important to have a "composite" description of the anaphoric relations. For example, in the "barber proof" on page <barber_proof> the barber *Giuseppi* is mentioned four times in the fifth sentence.

That is, he would shave himself if and only if [he] did not shave himself

An earlier version of MUMBLE which only looked at the immediately previous instance refused to use a pronoun at the point given in brackets, because the *if and only if* had created a parallel construction and the just previous instance of Giuseppi occupied [object1] and not the parallel [subject] position. The present version corrects this oversight by an *ad-hoc* modification to the feature-analysis procedure and the discourse history: whenever the next RECORD of a particular message element's discourse history is from a "less prominent" position than the previous RECORD (as in this example), this fact is noted and taken into account in subsequent feature-analyses; only special cases of "less prominent" have been implemented however.

4.3 Evaluating the pronominalization heuristics

Stated in terms of the computed set of abstract features, the actual heuristics are structurally trivial: they are simply checks for the presence or absence of certain features. Several of MUMBLE's heuristics are given below as examples. A heuristic either applies or it does not. If it applies, it contributes a "vote" into the pool; this vote may be either for or against using a pronoun, and at the moment may take one of three strengths: "necessary" (*white-ball* vs. *black-ball*), strong evidence (*strong-for* vs. *strong-against*), or weak evidence (*weak-for* vs. *weak-against*). Equal strength votes of opposite polarity cancel and ties are interpreted conservatively (i.e. against pronominalization).

```
(define-proz-heuristic introduced-in-subordinate-context
        vote strong-against
        condition (and part-of-qualifier
                       (not same-clause)))  ;i.e. we aren't in the same qualifier

(define-proz-heuristic pronominalized-last-time
        vote strong-for
        condition was-a-pronoun)

(define-proz-heuristic mentioned-once-already
        vote weak-for
        condition t) ;i.e.always true
```

Examples where pronominalization is ruled out are somehow more interesting than examples where it goes through. Below is a fragment of a description of a tic-tac-toe game (originally from [davey]). We want to look at the decision whether to pronominalize the second instance of the line "I" am threatening to complete—the decision-point is in brackets.

> *The game began with my taking a corner and your taking an adjacent corner. I threatened you by taking the square next to my corner and on the line opposite yours, but you blocked [it / that line] (...and simultaniously threatened me along the diagonal.)*

The relationship between the two instances triggers the following heuristics in MUMBLE:

```
mentioned-once-already  (weak-for)
not-most-recent  (weak-against)
introduced-in-subordinate-context  (strong-against)
```

The only vote in favor of pronominalization is the minimal one that is always present, and there two votes against it—one of them a strong one. Consequently, pronominalization is ruled out and another form of subsequent reference is selected ("*that line*").

4.4 Reasoning about distracting references

Except when the precede-and-command rule applies, linguistic relations alone are never enough to dictate whether or not a message element should be pronominalized.[20] Extranious referents in nearby regions of the text can lead to ambiguities for the audience when they are not considered in the pronominalization decision. The problem is, of course, that whether an ambiguity will actually occur depends on the semantic and pragmatic properties of the objects involved—properties that the lingustic component, by its nature, cannot know anything about.

---

20. While syntactic relations are certainly not sufficient, discourse relations (e.g. focus) might well be unless the speaker were using a very conservative strategy. This question should be investigated by testing different combinations of heuristics with a speaker/expert-program combination that is capable of producing large amounts of motivated text.

This is illustrated by the example below (from a hypothetical personal assistant program). Imagine that the linguistic component has reached the point given in brackets and must now decide whether to say "*her*" or "*Candy's*".

> "*Candy has come in late. Can Carol reschedule {her, Candy's} meeting for the afternoon?*"

Whether the pronoun would be ambiguous will depend on the particular audience and what they know. If they know that Carol is a group secretary and that Candy is a person who might call a meeting then they will correctly understand the pronoun because only one assignment of the relations will make sense. On the other hand, an audience that did not know who Candy and Carol were would be confused at best and more likely would make the wrong assignment because of the defaults.

Knowledge about group secretaries and meetings is much too domain specific to include in the pronominalization heuristics of an independent linguistic component. Instead, an appeal is made directly to the speaker program, via the interface function distinguishable-kinds. The arguments to this "oracle" are the message element being realized and set of possibly distractoring message elements (see below); its output is that subset of the distractors that it judges could confuse the audience.[21] In this example, distinguishable-kinds would return the empty set if the audience knew who Candy and Carol were, otherwise it would return the singleton set { Carol }.

When "speaker-certified" distracting references are present, the pronominalization routine presently employs a simple (if time-consuming) heuristic to "second-guess" how the audience would interpret a pronoun if one were used. Each of the distractors is run through the pronominalization heuristics as if it were the *current-instance* and their total votes compared. If one of the distractors has more heuristics in its favor than the actual *current-instance*, then it is assumed that the audience would take it as the antecedent for the pronoun and an alternative form of subsequent reference is looked for instead. In this example, the existing heuristics indicate that Carol would be the more likely antecedent because there are relatively stronger reasons to pronominalize it and consequently vote against pronominalization

Candy	*Carol*
not-most-recent (weak-against)	-------
mentioned-once-already (weak-for)	*mentioned-once-already (weak-for)*
------	*most-recent-np (weak-for)*
potential-actor-focus (strong-for)	*proz-able_current-subject (strong-for)*
------	*proceed-&-command (white-ball)*

21. In some applications, it might be pertinent to add the current message-level context as a third argument. This has not been necessary with any of the present micro-speakers.

Computing the set of "potential distractors" The extensive research on natural language understanding notwithstanding, there are no established formal algorithms for firmly delimiting the set of potential antecedents of a pronoun. Real texts can readily be found where in one case a pronoun is perfectly well understood even though separated from its antecedent by eight sentences, while, at the other end of the spectrum, a single interveening distractor will give the pronoun an unintended interpretation.[22] The best strategiem in this circumstance (and many others) is to experiment with many different heuristics and attempt to determine, by using the heuristics to control the production of many different discourses, which ones lead to the most natural results. Such experiments are still very much in progress; they are hampered at the moment by the lack of a (micro-)speaker that is both verbose and richly intentioned.

The present heuristics label as distractors any earlier references in the current sentence as long as they are not *relatively-subordinate*, the direct object ("theme") subject and verb phrase of the previous sentence, and the current focus of the discourse. This set is continuously updated by GRAMMAR-ROUTINES as the controller moves through the tree.

Before being passed to distinguishable-kinds, the set of potential distractors is filtered to remove those elements that can be distinguished from the *current-instance* on purely linguistic grounds. Broadly speaking, any element that is a distractor on positional grounds can be safely ignored if, were it the one to be pronominalized, it would result in a different pronoun than the actual *current-instance* will. This includes not only the obvious person, number, and gender differences (queried by the interface functions elmt-person, elmt-gender, and elmt-pluralp) but also more subtle grammatical effects. The reason why *Tremont* is not a distractor in "*Tremont didn't expect him up and running so soon*" is that had it been the antecedent, the TRANSFORMATION *equi* would have applied and the sentence would have read: *Tremont didn't expect ___ to be up and running so soon*".

4.5 Pronominalizing predicates

Predicates (more precisely: verb phrases) can be pronominalized on the same basis as noun phrases,[23] and using virtually the same decision process. The only difference comes in the definition of ignorable distractors, to which we can now add any message element that wasn't realized as a verb phrase. The actual "pronoun" used may be either the null-word (which will

---

22. A good discussion of this problem can be found in [hirst_pn_survey] section four]. He points out how even the strongest semantic restrictions can be (humorously) set aside apparently by just the recency heuristic, e.g.

If an incendiary bomb drops near you, don't lose your head. Put it in an bucket and cover it with sand.

This is a clear case of where **distinguishable-kind** needs to be sensitive to context, since while "bombs" and "heads" are have otherwise very different semantic properties, both can be "put in buckets". Note, however, that the possibility that the antecedant might be "*you*" never even comes up.

23. Clauses may also be pronominalized of course. However, the monologues of the existing micro-speakers have not provided any well-motivated examples.

draw "*do*" if the clause is specified negative or modal) or the phrase "*do it*". In MUMBLE, these two choices are treated as synonomous. An example of verb phrase pronominalization is:

"*Macbeth murdered Duncan because Lady Macbeth persuaded him to ___.*"

If this text were the beginning of a discourse, then there would be one distractor, "*persuade him...*", which, according to the heuristics discussed thus far, would be preferred over the intended antecedent because it is most recent. However, the situation is more complex than that as we shall now see.

'Self-containing antecedants' Generally speaking, there is no facility for recursive reference in a natural language: phrases must be finished before they can function as antecedants. This is why we can discount "*persuade him...* " as a potential distractor: if it were the real antecedant, it would have to be able to be read as "*Lady Macbeth persuaded him to persuade himself to persuade himself...*", which is clearly not likely.

The record of which phrases have been completed and which are still in progress is maintained as part of the discourse history. At the moment an elmt-instance is realized, its record is placed on a special list named pending-realizations. Only when the controller has finished traversing its realizing phrase is it finally removed from that list and added to the discourse history proper. In the example above, this means that at the point when the test for pronominalization is made, "*persuade him...*" will not even be visible. There are other circumstances, however, where a recursion of sorts is quite reasonable. One of these occurs when a message element has both a "label" and literal meaning, one example is labeling the formula that is to be disproved in a proof by contradiction "*the assumption*". There are typically several ways to express a label, some of which involve repeating the labeled element within the expression, e.g.

"*Therefore the assumption is false: there is no such barber.*"

In these cases, special measures are taken (described in section labels_n.y.w.) to insure that the correct entrys are used.

5. Discourse Predicates

Since its resurgence in the mid-eighteen hundreds with the discovery of IndoEuropean, Linguistics has undergone a steady progression in the size of the units it has considered for study. The very first work was done on the smallest units: phones and phonemes, with the scope of the analyses grew gradually to include morphemes and words; however, largely because of the enormous influence (in America) of logical positivism, it was not until the advent of Chomsky's generative grammar paradigm (1957) that any serious work was done on the syntax of the sentence.

Unfortunately, the subsequent twenty-odd years have seen little further progression: for too many linguists "grammar" remains synonomous with "sentence grammar"—we do not have a system of rules for discourses in any way comparable in richness or credibility with what we have for issolated sentences, nor is this situation likely to change soon. As a result, any rules of discourse grammar—larger than sentence-level constraints on decisions—that are proposed in conjunction with my theory of language production must be understood as speculative experiments that will probably require total revision as our appreciation of the correct primitives of discourse structure improves.

In this section we will look at three different kinds of problems: (1) how to define predicates that refer to relative positions within the discourse; (2) how to implement grammatical scope: constraints propagating leftward from certain words or phrases, especially quantifiers; and (3) how one would monitor for phrase-level ambiguities and whether one should. As always, the emphasis will be on the representational devices available to the designer and the trade-offs inherent in their use. The particular examples that will be shown have all been implemented in MUMBLE, but I have no commitment to continuing them should improved analyses become available.

5.1 Evaluating relative position

Discourse predicates are used to settle questions of style: Our high school English teacher tells us not to use the same adverb twice in the same paragraph, e.g. "*very*"; we translate this rule into a test that we apply every time we are contemplating using "*very*", in which we look back through the paragraph to determine whether we've used the word. That search (or perhaps just a direct lookup) is an example of a discourse predicate: if it is true, then we choose another word than "*very*"; if it is not, we do choose "*very*" (or maybe we do not, but for independent reasons).

'Happened-during'

The formalization of a deliberation like this revolves around a record—the "discourse history" pg.<the_discourse_history>—that associates *events* (e.g. msg-elmts realized, grammar-decisions made, choices selected) with *regions* that are identified in terms of relative position from the controller. Examination of this record fundamentally involves the same kind of test in every case:

(happened-during <event> <region>)

where happened-during returns true or false. In specific predicates we might also need to know some of the details of the events, however, the most important information will remain "how long ago did this happen", since it is the primary determinant of the relevance of a past event for current decision-making. This kind of information is the same as we saw being used to decide whether to pronominalize, the only difference being that now we will ask a broader range of questions about both events and regions. Let us look at the general problem of computing happened-during and then at a real example from MUMBLE.

Since each region will invariably contain many events, efficiency considerations (i.e. employing associative lookup rather than search) dictate that happened-during access the record of the event first and use that record to identify the region it occured in, rather than first accessing the event. The record of an event is compiled at the time the event occurs (section record_n.y.w.) and then associated with the generic form of the event (the msg-elmt or the decision or the choice) as part of its history. Within the record, we include the regions in which the event occured, identified by their absolute position (e.g. from the start of the discourse).[24] In MUMBLE regions invariably corresponded with constituent structure nodes, making the node's record (pg.<node_n.y.w.>) the obvious object to use to represent the regions in the discourse history.

While regions are recorded in the discourse history in absolute terms, the discourse predicates—being, as they are, part of general heuristic rules—must refer to regions in relative terms that they can be sure will always be applicable. To this end, we have the controller maintain a set of controller-variables that pick out the regions we are interested in, and have them updated by grammar-routines as the controller moves. MUMBLE, for most domains, kept track of the following variables (in rank order by size): previous-conjunct, current-sentence, previous-sentence, current-discourse-unit, previous-discourse-unit, current-paragraph, and previous-paragraph.

---

24. It would not be possible to maintain the alternative scheme of recording the identity of regions by their position relative to the controller. As the controller continually moves into new regions, all the old records would have to be edited to correctly reflect the new relationships between the events and the controller. In the course of realizing a message, this editing and reediting would be happening continually, consuming exponential time, and thus the alternative is counter to the stipulations of the theory.

The values of these controller-variables are always the records of the nodes that defined the regions—the same records as are used to define the regions where events occured in the discourse history. Thus to test happened-during we retrieve the record of the event, extract the name of the size region we are interested in, and compare it with the value of the controller-variable given as the argument to be tested against.

'which is' versus 'leads to'

In designing the dictionary for the logic domain, I had to decide how to realize the case where a formula in the proof denoted a contradiction. Following Chester's lead [chester], I elected to have the formula expressed as a meta-comment about the proof. That is, rather than expressing a literal relation over its subelements (e.g. "*he both shaves and doesn't shave himself*"), it will express a dependency relation between some earlier formula and this special event in the proof. (In natural deduction proofs, each line is annotated with the inference rule used to derive it and the numbers of the earlier lines that the rule referred to in making its deduction.)

Two phrases came to mind as ways to express this relation, and the relevant decision criteria involved the structure of the discourse. They were treated as a marked-unmarked pair: in the unmarked case, the phrase was:

"*<earlier wff> leads to a contradiction*"

In marked case, where the earlier formula was "the source of the previous sentence", a special syntactic manouver was made and the phrase attached as a relative clause:

"*<sentence for earlier wff> which is a contradiction*"

Testing whether a candidate formula is "the sourse of the previous sentence" is an instance of the limit case of happened-during. We start with the earlier formula, say conj102, and look up its last record in the discourse history. From the record we extract the sentence field of its position property (example on pg.<macbeth_discourse_record_n.y.w.>); it will be the record of some sentence-level clause node. To answer the predicate, we test whether this record is the same individual as is currently the value of the controller-variable previous-sentence.

Structuring the Discourse History

A little introspection into one's own writing style will be sufficient to reveal that a "finer grain" is required for the discourse that just noting sentences and paragraphs. In the pronominalization routine, for example, MUMBLE makes its decision about what the potential distractors are by looking within the previous sentence at the message-level sourses of the direct object (theme), the subject, and the predicate. Rather than point to these with individual controller-variables that are unlikely to be used anywhere else in the grammar, we make them part of the record of the clause that contains them, and have the pronominalization routine's test proceed by an indirect route via that record.

Similarly but on a larger scale, we have the paragraph structure used in the KL-ONE-nets-as-objects domain: I experimented there with having references made in the first or last sentences of a paragraph "decay" at a much slower rate than references elsewhere in the paragraph. The effect of the difference was felt by the predicate that determined whether a msg-elmt was "given". The working definition of given was "mentioned but not yet stale" (i.e. not yet forgotten; see section given_new_n.y.w. for the full discussion), where the boundaries of when a reference became "stale" were now tied to paragraph structure.

To support this experiment, the record for a paragraph was given two properties: topic-sentence and last-sentence, whose values were the records of those sentences. The properties would be entered into the record as the sentences were completed, using paragraph-level grammar-routines to do the processing. The record for the first paragraph of the text on page <given_klone_example_n.y.w.> is shown below.

Para1		
topic-sentence	Clause2	
	subject	phrase
	theme	top-of-the-net
	predicate	(superconcept X top-of-the-net)
final-sentence	Clause24	
	subject	phrase
	theme	conjunction27

In determining whether a msg-elmt is given, we use this record from the bottom up as it were: first we define the region relative to the controller within which we will say that a reference will still be remembered (i.e. "is not yet stale"); then we look to the history of the element we are interested in and compare the name of the sentence that contains it with the names of sentences in that region. In the experiment, a msg-elmt was "given" if:

(1) The controller was in the first sentence of a new paragraph and the last reference to it appeared either:
   (a) in the last sentence of the previous sentence, or
   (b) in any of the earlier topic sentences;
(2) Or if the controller was in body of the paragraph and the last reference appeared anywhere within that same paragraph.

This scheme produced satisfactory results on the material of the KL-ONE nets. However this material was very uniform (it rapidly became boring to read), and, in particular, it never was the

case that a reference from within the body of a paragraph (i.e. in neither the first or the last sentence) was ever repeated in later texts, thereby making the definition of given adequate tautologically.

Synonym-set

The most sophisticated use of the discourse history that had appeared thus far in MUMBLE is the decision procedure synonym-set: a decision-extension that may be used whenever the designer has a set of choices for which s/he can find no distinguishing criteria other than a variation for variation's sake. A good example of such a set might be the class of phrases George Lakoff termed "hedges" [lakoff_&_gordon_hedges], phrases that we use to temper or soften our statements (as in "hedging a bet"). For example:

"Professor Winston will probably be busy all afternoon."
"I think that Professor Winston will be busy all afternoon."

At least at first glance, these phrases mean the same thing, and as designers we would like a uniform routine that will capture this synonomy.

Synonym-set is a share-able decision that will randomly select one choice from the set given it as an argument, subject to two conditions:

(1) The same choice is not selected twice in a row.

(2) In coordinated contexts the same choice is made in every conjunct.

Within an entry, synonym-set is written as though it were a choice, e.g.

```
(define-entry  <some entry>
   <earlier DECISIONs that construct the matrix>...

   (hedge   ; the decision-name
      default  (synonym-set  (probably_post-aux-adv  matrix)
                             (speaker-thinks-that_pre-sentence-adv  matrix))))
```

Formally, however, synonym-set is a decision—a critical fact since the results of decisions are kept in the discourse history and thus we do not need a special device just to record the activity of synomym-sets. We satisfy the first condition by referring back to the record of the most recent occasion when the synonym-set decision was made and seeing which choice it made then and thus which choice to avoid this time.[25] The second condition is handled similarly, testing for whether a previous instance of the decision occurred in within the region defined by the controller-variable

25. The careful reader will have noticed that there may well be several instances of the synonym-set decision in the discourse history but with different sets of choices. This potential ambiguity is forstalled by the expedient of having the load-time postprocessor give each instance of the same set a unique name. The postprocessor is also used to construct the actual decisions by expanding fixed schema so as to customize them to, e.g., the number of choices in the set, and thereby make the computations more efficient.

previous-conjunct, and this time making the same choice.

5.2 Scope

The intriguing phenomena of the scope of grammatical operators has been given short schrift in this thesis. The blame, if that is the right word, lies with the choice of domains for the speakers and expert programs, since none of them had naturally motivated occasions to use scoping operators; in particular, none of them made any non-trivial use of quantification. We have seen in other sections how the details of the message-level representation have deeply influenced the ways in which linguistic analyses should be made; I accordingly elected not to look at phenomena for which there was no independently derived representation in the domain.[26]

Two small cases of scoped operators did come up in MUMBLE however, and they may be sufficient to indicate the way in which the problem of scope could be delt with: these are the "*some*" to "*any*" "transformation" in the presence of "*not*", and one case of attributive adverbs.

'some' versus 'any'

It has been observed at least since the time of Klima's important paper on negation [klima] that in certain enviroments the quantifier "*some*" can not be used and that "*any*" should be used in its stead.

"John has some money." "John doesn't have any money."
"John is too poor to buy anything (something)."*
"If John has any (some) money, he's luckier than I am."*

This generalization has many "holes" in it however, as neatly pointed out by Robin Lakoff [lakoff_some_any]: it often *is* preferable to use "*some*" in those context, and the beliefs and presuppositions of the participants involved in the speech-act must be taken into account in order to make the decision. Consider the total difference in intent of these two statements that might be made by people at a picnic upon seeing a chipmunk come up to the edge of the blanket: (Example due to Terry Winograd.)

"If you don't give him some food, he'll go away." (i.e. Give him the food!)
"If you don't give him any food, he'll go away." (i.e. Don't give him the food!)

Still, for all its faults the rule remains accurate in many specific cases. The one that turned up in MUMBLE involved the use of an automatically generated pronoun to stand for the case of the generic individual.

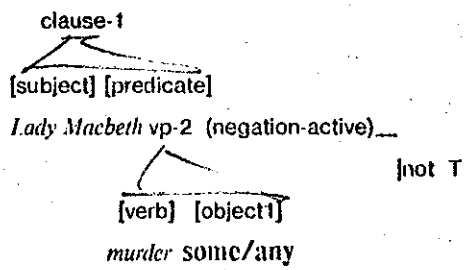
---

26. This is unfortunate since the problems presented by quantifier scope are among the most complex that people still manage to deal with successfully. The question of what kind of planning and "lookahead" one needs in order to insure that one has conveyed one's intended meaning in a text like "*Some Eastern Airlines flight goes to every city in the Carribean*" is very intriguing.

"There is some barber who shaves everyone who doesn't shave himself."
"There isn't any barber who... "
"Lady Macbeth did not kill anyone."

In order to accomodate this context-dependent variation, the message-level precursors of these generic individuals: quantified variables in the logic domain and wildcard matching variables in the Macbeth domain, are realized by their dictionary entrys as what is in effect another msg-elmt, call it some/any.[27] The point of setting up this extra "level" between the speaker's quantifier and the English vocabulary is to decouple the basic realization decision to use a pronoun and the grammatically motivated decision (or "reaction") to being in a negative scope; as the designer, one could not guarentee that the scope information would be available at the point where the decision is made, nor should one try to since the two decisions are of very different sorts and ought not be combined.

If the boundaries of the scope of operators like "*not*" correspond to constituents, then the "being in the scope of" predicate can be readily modeled by a region-feature. An action is added to the grammar-routine that enters the operator into the tree; its effect is to add the feature, say negation-active, to the node for the appropriate constituent. The presence of that feature is then detected by affected decision-makers such as the entry for some/any. This situation is shown in the snapshot below, taken just at the point when some/any is about to be realized.



"Lady Macbeth didn't murder... //"

Everything within the region dominated by verb phrase two (the shaded region) is in the scope of the negation. (A separate feature is used, rather the *neg* hook which already exists, because we presumably want this triggering condition to apply to other contexts besides an explicit "*not*", such as hypotheticals or questions.) When the entry for some/any (below) is interpreted, it looks

27. Some/any really is a msg-elmt in that it is a valid argument to the interface functions (i.e. it has an entry, has a name, has a reference-type, and so on). At the same time, of course, the representation used to implement some/any wouldn't necessarily be the same as the expert program's and thus could require some designer tinkering with the interface function definitions.

for the presence of the feature, and selects "*any*" if it finds it, otherwise "*some*".

```
(define-entry some/any_entry ()
    default (use-word "some")
    ((is-true 'negation-active)
     (use-word "any")))
```

The scope of the 'Opinion-holder'

Perhaps the most rewarding moments in doing this kind of research come when your rules "break" because you've stumbled across a fact about the language that wasn't obvious before. They are particularly valuable when one is faced with a set of apparently "synonomous" phrases and is trying, because of a prejudice that complete synonomy does not exist, to find the contexts or the intentions that set them apart from each other. One such moment occurred with the earlier synonym set: "*I think that...*" versus "*...probably...*". These two phrases can be categorized functionally as "attributive adverbs": they are used to bind the opinion expressed by a proposition to a particular person, in this case the speaker. On their face, they both serve this function equally well, which is why I originally encoded them as a synonym set. However, it turns out that there is a kind of context in which the two are not equivalent, and the difference turns on a matter of scope. Consider these texts, taken from some preliminary work with a program that modeled commodities transactions [sr_reading_commodities_paper].

"According to Grain Market Review, *wheat will be cheaper."*
"I think that according to Grain Market Review, *wheat will be cheaper."*
"According to Grain Market Review, *wheat will probably be cheaper."*

The second and third sentences were produced by applying the synonym set to the proposition given in the first, with the intented goal of "hedging" the speaker's opinion (perhaps they only skimmed that issue of *Grain Market Review* and are not really sure of what it said). Notice however that the third sentence does not hedge the speaker's opinion but the magazine's!

It turns out that there is something like a "current opinion holder" present as part of the linguistic context of every text: propositions that are matters of opinion are attributed to this authority unless the text contains explicit instructions to the contrary. The default assignment of the "current opinion holder" is typically to the speaker, but a phrase like "*according to...* " will reassign it to the indicated authority (in this case the magazine), at which point the new assignment stays in effect until there is another explicit change or that discourse unit comes to an end.

The implementation of the hedge "*I think that...* " (pg.<I_think_that_adverb_n.y.w.>) always puts the phrase at the beginning of the sentence, and thereby effortlessly avoiding any confusion introduced by any subsequent attributive modifiers to the sentence. The implementation of "*possibly*" (pg.160), on the other hand, is done with respect to the main verb of the clause (e.g. "*be cheaper*"), with the effect that something analogous to "variable capture" in programming languages will occur and the speaker's intent will be thwarted. The "fix" to this problem is (1) to augment the English grammar by introducing a controller-variable for the current opinion holder along with managing grammar-routines to be associated with the attributive modifier function as appropriate; and (2) to remove the two hedges from the synonym-set and reinstate them as a regular decision with a explicit check that the preconditions for the effective use of "*probably*" are met.

5.3 Detecting structural ambiguities

As a linguist sees them, texts are grouped into larger and larger units organized into a tree structure. This is also true for the speaker (at least given the present model) since it is exactly that tree structure that is being used to control the whole production process. Unfortunately however, when a person is reading a text the units are not visible but must be deduced, and there is no guarentee, given the inherent ambiguity of natural language texts, that the reader will deduce the same tree structure for a text as the speaker used in producing it. (This is not generally true of speech: when a text is being heard, the pauses and intonation contours are usually sufficient to mark the units explicitly.)

The lack of any guarentee that the audience will parse a text in exactly the way that he produced it poses a quandry for the speaker: how much effort should be devoted to avoiding unintended ambiguities, and where will it be best placed? I will argue here that very little effort can be devoted on-line because of an intrinsic limitation deriving from the fact that at the time when a potential problem could be noticed it will nearly always be impossible to do something about it. Furthermore the overhead of burdening the linguistic component with monitors for these potential ambiguities is very high; were I already in possession of a theory of how the speaker's *own* comprehension system reacts and feeds back to the production process, I would present it here and argue that the task of detecting a structural ambiguity is done best by a device that is suited to it—a parser that buffers many constituents at one, rather than a generator with a single point of action. However I do not have such a theory (though I hope to develop one).

Consider this sentence, which I wrote (and then edited) in preparing this document:

> "*Unless preplanned, sentence-chunking decisions are made at each "potential-sentence-start" as defined by the grammar.*"

In the parsing literature, that sentence would be referred to as a "garden path": All the while following a perfectly normal parsing strategy, the reader will interpret "*preplanned*" as an

adjective modifying "*decisions*" when in fact I intended it to be the end of an introductory phrase. The reader mistakenly extends the introductory reduced clause "*unless preplanned*" through the rest of the sentence, fails to find a main verb, and becomes confused. The "fix" requires rephrasing the introduction so that its function is unmistakable:

> "*Unless they are preplanned, sentence-chunking decisions are made at each "potential-sentence-start" as defined by the grammar.*"

If we were going to write a monitor to "catch" such an ambiguity while still staying within the theory's stipulations (e.g. no simulation and backup), then we simply could not arrange to use the fix that I used in the text. The trigger condition for the monitor would be the occasion of an introductory adjunct ending in an adjective, followed by a noun phrase in the [subject] which could reasonably be modified by that adjective. The critical problem is that the fact of the adjunct terminating in an adjunct can not become known until after the adjunct has been completely produced; a monitor stationed as, e.g., part of the [enter-slot subject] grammar-routine would not be triggered until it was too late to make the obvious edit, and the ambiguity would go through.

One can imagine how refinements to the way the linguistic components were described might make it possible to *forestall* the ambiguity by detecting the possibility that it might arise before any commitments to specific texts had been made. If, for example, we had a functional characterization of the introductory adjunct described it as attributing some property to the subject, then at the time that the adjunct's message-level precursor was positioned in the tree we could predict the possibility of the ambiguity and proceed to influence the amount of ellipsis that the adjunct could undergo.

This technique can be more difficult for the designer that using a monitor might have been: it requires careful additions to the conceptual base of the grammar, and the undertaking of a proper analysis to insure that the additions are suitably general. However the rewards are worth the effort since the freedom of action available "above" the site of a potential ambiguity and before any of the text has been realized is enormously greater than that available "to the right" of the sight after most of the relevant text has already been spoken.

6. Reasoning about possible CHOICES

A subtle but telling point of difference between the present design and the alternative school of "production directly from program data" exemplified by [swartout_masters][chester] [roberts_direct_production] is the way arguments to a message level relation are treated. Direct production schemes invariably follow a strict depth-first evaluation protocall, i.e. they compute the realization of the arguments before the realization of the relation itself. The present design reverses that order, realizing the relation before its argument, a technique that is sometimes referred to a *delayed binding*. The evaluation of the relation creates a program (expressed as a surface-level, English constituent structure) with places for the evaluation of each of the arguments embedded within it.

The advantage of this form of delayed binding is that individual arguments are not realized until the linguistic context in which they will appear has been fully created and is available to influence and constrain the computation. At the same time however, the decision-process that realizes the relation is now denied the "precognition" available in the direct production scheme: it cannot know what English phrases will be selected for its arguments—at least not as a literal text string or constituent structure. This is where the ability to symbolically analyse dictionary entrys comes into play.

One of the places where his ability is particularly important is a conceptually sparce domain such as the logic domain (VII.B.2). Here the speaker's goal is to always use as "fluent" a construction as the logical expressions will permit. The *implication_entry* (pg. 205), for example, includes the test:

```
(A-is-a-subject-for-predicate-B  antec conseq)
```

If this test succeeds, then the ENTRY has license to use the most fluent construction it knows of, namely mapping the antecedant and consequent into the [subject] and [predicate] constituents of a single clause (as in "*All men are mortal*"). If the test fails, then the ENTRY must fall back on more literal constructions such as "*A implies B*" or "*if A, then B*".

Generally speaking, answering a question like this requires search. Specifically, a search that chains backwards from a desired realization, through the decision-rules that select it, to the present state of the tree and discourse-history—the determinant of which of the possible chains of predicates are (or could be) true. The linguistic component is not designed to support this kind of search in its most general form because it requires a non-deterministic control structure, leading to a process that more resembles hill-climbing than the one-pass refinement of a tree. However, certain limited forms of this search can be precomputed once and for and then used without encurring additional costs at run-time. ENTRYs can use this precomputed information essentially as a sort of "linguistic type-checking" to help them decide between alternative realizations.

6.1 The technique in brief

The schematic design of a dictionary ENTRY makes it relatively straight-forward to compute a symbolic answer to questions of the form "what kinds of realizations will this ENTRY choose". (Indeed, that was one of its design criteria.) At load-time, the postprocessor computes an inversion of the *matrix* DECISION[28] of each ENTRY—I will refer to such tables as choice-inversions. They have the following fields:

(1) a summary of the linguistic properties of each CHOICE

(2) the choice-application itself (for the few occasions when more detailed information is needed),

(3) a list of the predicates that must be true if that CHOICE is to be selected and not some other.

(The inversion includes the modified CHOICEs created by TRANSFORMATIONS. It consequently is quite large and is specially designed for maximal sharing and efficiency.)

The most common questions, e.g. "will the ENTRY select a noun phrase given an ELMT-INSTANCE in a nominal context?", can be answered quickly from the pre-computed summaries. More elaborate questions, such as "for a given ELMT-INSTANCE will it select a clause whose [subject] is, e.g., man(x)", can have their answers computed on demand by a selective symbolic evaluation of the CHOICE. This results in a set of candidate choices, each of which is then checked for plausibility by testing if the predicates it is paired with are or will be true. This is the potentially unbounded step of the procedure: when the selection of a choice is contingent on facts that are not already known (such as how one of the elmt-instance's subelements will be realized), those facts must be expressly computed. If the new computations in turn require still futher computations and assumptions, then the process begins to look like forward simulation rather than precomputed lookup.

6.2 Will-be versus could-be

At this writing, reasoning about possible choices is done in only two places: in the entrys for the logic domain, and in the shopping-list entrys of the tic-tac-toe domain and the real KLONE domain. In the logic domain, it is a necessity if the more fluent constructions are to be selected only where they are appropriate, but the extreme "narrowness" of the predicate calculus formulas and the many contingencies of their entrys can make the search very deep and very broad. Shopping-list entrys, on the other hand, are used entirely in domains where the entrys for the kernal propositions being tested are always simple and the tests can be made on the basis of a "will-be" based analysis rather than the less bounded "could-be".

---

28. To include any of the contingent or refining decisions in the invertion would add combinatorically to the size of the final structure without adding any information that it is useful to anticipate(at least that is the experience at this point in the research.

Trivial entrys, such as the ones that were written for the Macbeth domain, consist only of a default choice, and thus have an entirely predictable output. The choice-inversion of such an entry will always be found under the entry's will-be property. Reasonably enough then, the could-be property is reserved for entrys with multiple choices. (In MUMBLE at the moment, none of the heuristics that look at this predictive information need to know anything more specific than a phrase's category. This permits the simplification of treating multi-choice entrys where the choices all involve realizations of the same category as though they were single-choice and using the will-be property.)

Corresponding to the two cases are two different predicates for actual use in the decision-rules of the entrys.

(will-be <elmt-instance> <category>)

(could-be <elmt-instance> <category> <conditions>)

Will-be can be evaluated just by comparing the given category to the one given in choice-inversion of the elmt-instance. Will-be is consequently quite inexpensive—it is could-be that can initiate the potentially unbounded search.[29] If could-be is true, it returns the set of all those choice-applications that satisfy the conditions, otherwise it returns *nil* to indicate "false". Will-be just returns *T* if it is true or *nil* if false.

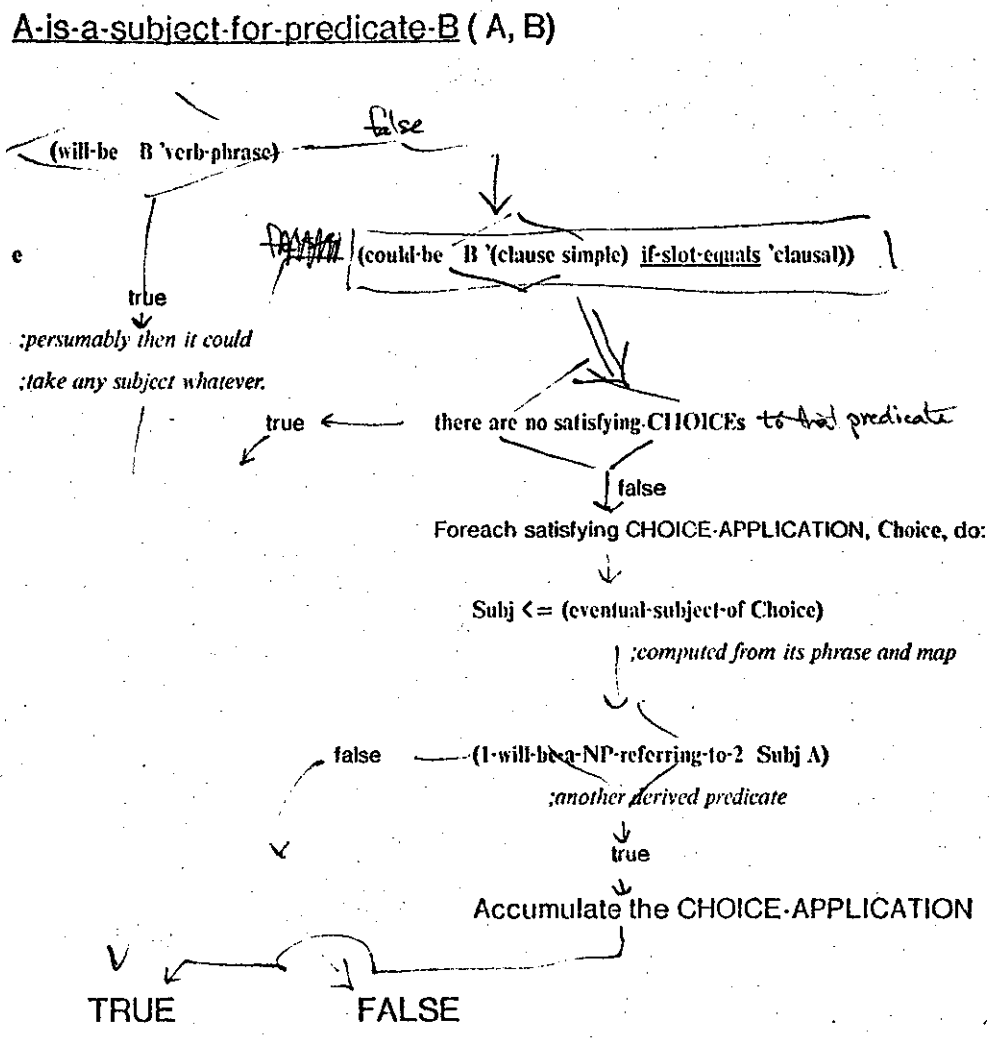
At this writing, the notion of just what can be a "condition" to could-be is not settled. It must be some property of the linguistic context that the entry making the test either knows is true already at the time the test is made, or one that the entry is in a position to bring about. The only condition actually used in the dictionaries of the micro-speakers was the specification of the slot in which the elmt-instance will be contained; conceivably however, it might be useful to also use specifications of the values of various grammar-variables or of attachments that the elmt-instance involved will have.

6.3 Derived predicates

A "derived predicate" like A-is-a-subject-for-predicate-B is defined in terms of the basic predicates will-be and could-be. At the moment, there are no limitations or common patterns on the kinds of control structures that derived predicates may employ and consequently they are couched directly in LISP as procedures. The flowchart below is typical. (A and B are elmt-instances.)

---

29. Strictly speaking, the maximum number of entrys that could ever be searched must be less than or equal to the size of the branch of the enumeration order that starts at the msg-elmt in whose future we were originally interested. The only problem comes when we have to ask the same question several times while traversing the same branch (which does happen in the logic domain). If there is a scheme to remember the forecasts as they are first computed--a well-formed-substring table in effect--then the time requirement can be kept to polynominal.



An example of how deep a analysis will go Below is the well-formed-formula that constitutes the first line of the barber proof (pg. <barber_proof>).

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

In an unmarked discourse context, it is realized as: "*There is some barber who shaves all and only those men who don't shave themselves*". With the existing dictionary of the logic domain, the selection of this text requires the evaluation of nine basic predictive predicates in four different derived predicates, recursively embedding the analysis down five levels, and eventually reaching reaching all of the leaf predicates and constants of the formula. Here is trace of the process, followed by a prose description.

The reason for this depth is not difficult to see. The realization of the existential quantifier as a "*there*" clause depended on being able to realize the conjunct as a noun phrase; that in turn depended on (1) both of its clauses being realizable as saying something about the same variable (""X"), and (2) the first clause being realizable as a name for that variable. Evaluating the second criteria simply required a one-level lookup—could the predication barber(x) lead to a noun phrase naming x? (answer "yes")—but evaluting the first cirteria lead to a recursive analysis, i.e. it required determining that the second clause, the universal formula, could be realized as a clause (with the variable as its [subject]) leading us to a recursive examination of formula-entry (pg.207). The realization of the universal formula depends on that of the biconditional, i.e. to meet the criteria it must be expressible as a simple clause, which will happen only if (1) its antecedant subexpression can be a clause, and (2) its consequent can be interpreted as a restriction on the range of one of the variables in the antecedant, in which case the consequent can be embedded within the realization of the antecedent as a restrictive relative clause on that variable. (The predicate that tests for this condition is referred to as merge-var.) The (local) first criteria depends on whether the predication that makes up the biconditional's antecedant can be realized as a clause, which depends on what the predicate involved ("shaves") will be in a clausal context. In this case it will be a clause, giving us the first case of a test in this chain that yielded a direct yes-no answer and not a recursion. Similarly, the merge-var text gives a direct answer (it looks at the patterns of variables within the expressions and does not ask linguistic questions). Finally now we can unwind this stack of functions and ultimately answer yes: the there clause can be used.

Limitations on symbolic analyses As a practical matter, this kind of analysis can become quite involved: Once one begins to apply predictive linguistic predicates to the subelements of subelements, the search space becomes awkwardly large, taking considerable time to explore, and the amount of simulation involved becomes excessive to maintain without elaborate overhead.

On a more theoretical level, any symbolic exploration of how the tree will be (or could be) refined from some point will be fundamentally limited in the kinds of information it can find out unless it involves a real forward traversal by the controller (and thus the production of real constituent structure). Only the controller is able to notice and execute the grammar-routines associated with the linguistic properties of the tree and only the controller is able to utilize the subsequent reference as well as the main stream routes through the realization procedure.[30] With out the controller then, there will be no information about function words, morphology, many elliptical phenomena, readjustment rules, verbal auxiliaries, gaps due to wh-movement, or (and this last is particularly important) grammatical constraints on the realizations.

30. This constraint is nearly tautological since those two abilities are defined to be part of what it means to be the controller. Only the depth-first left to right traversal pattern, the extension of the tree by substitution of realization for realizee, and the actual printing of the words were left out. However, they too must be included when you consider that the grammar and discourse heuristics were written with the expectation that they would be part of the process; consequently the grammar-routines would be likely to produce subtly wrong actions were any other conventions used.

"Predictions" arrived at by symbolic analysis of entrys can be believed only to the extent that they do not depend on controller-based linguistic information. In general this means that only very small portions of the enumeration order can be analysed reliably, perhaps no more than one msg-elmt and its subelements. The enormous analyses of the logic domain succeed as well as they do because the reasoning its dictionary does is not very dependent on linguistic context; It is possible that certain narrow, grammatically defined regions of the tree may be reliably "predicted across". Once the controller is entering the [subject] for example, it is probably "safe" to look ahead to the msg-elmt in the [predicate] and predict what main verb will be used. Even there however there are possibilies for unforseen interactions if, say, a length relative clause or apposative interveened.

One can only be absolutely certain of decisions that have already been made and on future decisions that depend *solely* on those decisions. This is not a small number since decisions that specify discourse-level features such as focus or given/new can have a very pervasive effect on decisions within their regions. The number and subtlety of discourse-level specifications that will be in use will greatly increase as our experience with production grows; effective predictive ability then may come to rest not in syntactic level simulations that are guarenteed to be incomplete but in detuctions from established discourse level facts.

CHAPTER FOUR

DICTIONARY DESIGN

Dictionary design has three aspects:

(1) developing the interface between the representation of the new domain and that of the linguistic component

(2) writing the actual entrys: deciding on the appropriate way to decompose expressions from the domain and customizing the linguistic component's choices, looking for modularity in the decisions

(3) taking advantage of, and extending, the interlingua: the conventions and subroutines of the dictionary that can be shared between domains.

This chapter devotes a section to each of these aspects. As before, the emphasis will be on the problems and the trade-offs among their solutions. We will not look at an entire dictionary (the specific problems of specific micro-speakers have been put off to appendix VII.B.1.3) We will look at individual entrys, each drawn from the micro-speaker that provides the best example of some generally-applicable technique. The details of any single dictionary are dictated the particular ontology and representation of that domain—the dictionary for, e.g., the logic domain will tell us more about logic than it will about dictionaries. In addition to these problems of parocialism, the actual dictionaries are all experimental, i.e. new techniques were continually under development and there was no attempt to enforce consistency upon old dictionaries as the design matured.

1. Issues at the Interface

If the linguistic component were being used with only one speaker and expert program then there would be no need for an interface:[1] all of the speaker's conventions and the linguistic component's requirements could be integrated, and the responsibility for maintaining them distributed evenly across a seamless merger of the two decision-makers. An explicit message (with its implied message constructor and message interpreter) would be required only when it was necessary to represent goals that referred to it (e.g. "don't be long-winded" or "don't use technical vocabulary"). But of course the opposite is true: linguistic component is intended as a separable module, computationally as well as conceptually, and a uniform interface is thus required to smooth over the differences between domains.

This section is concerned with the conceptual and technical requirements on the interface and the problems they derive from. It looks at what kinds of access are required of the common computational enviroment, and divides message elements into three categories on the basis of their relationship to the other data structures of the speaker and to their entrys. The highly important question of how much specification the speaker can or should include in their messages rather than leave to the defaults of the linguistic component is then raised, but answered only pragmatically. Finally, the technical question of exactly how a message element—something that the speaker's dictionary knows how to describe—can be identified as such, and what the "lower limits" are on that process.

1.1 Bridging the gap between modules

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 kinds of things will want to be said or how they will be expressed. Instead, the speaker must construct an explicit description of what it wants done (i.e. the message), and must pass it to the linguistic component along a well defined channel (pg.<initialization_n.y.w.>). Let us look at some of the contingencies that govern what messages can be constructed from.

First of all, we must appreciate that ultimately the designer is allowed to structure messages in anyway she or he pleases. This is due to a fundamental "out" designed into the framework of the interface, namely that the enumeration order of a message—the "glue" between its elements—is determined by the access-functions that the designer chooses to use in the dictionary and not by the controller or any other fixed part of the linguistic component. It is dynamically

1. It is very likely that the whole idea of a functionally defined interface has no relevancy to the human "implementation" of language production. There would remain, however, a fundamental difference in the kinds of relations and object that had to be represented in the human equivalents of the speaker and the lingusitic component and thus some kind of physical separation and interface remains plausible as a hypothesis.

interpreted rather than set down once and for all time. Since we cannot now anticipate all of the kinds of representations and system configurations that future designers may want to use, we should provide this "out" and define the form of the interface as flexibly as we can: by functional specifications rather than structural ones. Except for a kernal of technical requirements (described at the end of this section), the designer may use virtually any conventions they wish in bridging the gap provided that the conventions of the dictionary are matched to them.

Computational environment

Making decisions that are dependent on context requires (1) having an examinable representation of the context, and (2) being able to test predicates that refer to it. The computational process of maintaining and updating the representation and of evaluating the predicates can be limited in its accessibility. If we consider all of the information that a process can access at a given moment, call this its *computational environment*, then we can picture this environment as being segmentable—parts of it may being added or taken away as time passes with the process's ability to access information changing accordingly.

The speaker/expert-program and the linguistic component live in disjoint computational environments: they know different facts, they employ different representations, and they solve different kinds of problems. (That is the only reason why they can sensibly be different modules.) The dictionary, on the other hand, is a different matter: it by definition is the place where the two modules come together; the decisions made by the dictionary typically involve accessing both computational environments. Accordingly, the relative configuration of the two environments can matter a great deal.

The linguistic component may or may not be in the same computational environment as its domain. (Among the micro-speakers, logic, Macbeth, and tic-tac-toe shared the environment, while the Digitalis Advisor and KLONE-nets-as-objects did not.) When environments are shared, the speaker may construct the messages directly from the expert's data structures (possibly adding structures of its own). This is the ideal design, and has made for the technically simplest messages and interface functions. When domain and linguistic component are independent programs, some artificiality must be added: the speaker's message must be converted into some external form (e.g. a text string) and then parsed when it reaches the linguistic component; similarly, the interface functions must apply across the gap, requiring an external form by which to name message elements. But while separate environments may complicate the interface technically, they do not force conceptual changes in the elements as they cross the gap.

Message-element Categories

We can classify message elements (and by extension messages) into three categories according to their relationship to their realizing texts: "issomorphic", "shopping-list", and "arbitrary". *Issomorphic* message elements are composites and have a structural correspondence with the phrases of their texts. *Shopping-list* message elements are pointers—names—that can be used to access a conventionally organized body of facts about the element; their entrys consult a speaker-supplied or default list of these facts to determine which should be used in a given realization. *Abitrary* message elements have no structural relationship to their realizations: they are a device to pick out phrases that the designer thinks should be used to express some state of the domain. We will look at each category in turn, drawing on cases studies from earlier programs for examples.

Arbitrary message-elements These derive their name from the fact that none of the properties that they may have either for the speaker or the expert program have any effect on the way the linguistic component realizes them. Their entrys do not decompose them nor do they use them as a way to access other objects from the domain. Arbitrary message elements are such either because they are primitives in the domain—atomic names are a good example, or because they are being used as a stop-gap measure to identify some condition or state that the designer would like the speaker to be able to talk about but for which the program does not yet have a name or representing data structure.

Winograd's SHRDLU program [shrdlu] provides a good example of an arbitrary message element. SHRDLU had a "failsafe" heuristic for the resolution of pronoun references: when it was not certain of its user's intended antecedent, it would anounce the assumption it was making by saying:

"*By "it", I assume you mean the block which is taller than the one I am holding.*[2]"

SHRDLU did not have an explicit representation for the "mental state" that prompted these announcements, i.e. there was no assertion such as (assuming-antecedent "*it*" block1) ever added to its data base. Instead, the semantic specialist within SHRDLU that made the assumption initiated a procedure call directly to the answer routines, with the pronoun and assumed antecedents as arguments. Thus the only representation of the assumption and the intention to communicate it was a transitory pattern in the program's flow of control.

If we were going to reformulated this part of SHRDLU's output in terms of MUMBLE, we would have to do very much the same thing. Triggered, perhaps, by the same pattern of procedure calls, would be a routine, conceptually part of the speaker, that would be responsible for setting up the message and sending it to MUMBLE. To construct the message, it would need

2. SHRDLU was not capable of producing a relative clause on its own. This one is a paraphrase constructed from the user's input sentence.

some object(s) for MUMBLE to interpret, but since no existing object that already stands for the assumption (i.e. it is only a process state), the message-building routine would have to make one up. It doesn't matter what kind of object the routine chooses as long as (1) the object is linked to the entry that knows how to say "*By X, I assume you mean Y*", and (2) some provision is made for the two argument subelements. A list such as (G006 "*it*" block1), would be perfectly adequate. G006 would have no meaning anywhere except in MUMBLE's Blocks World dictionary, where it would be interpreted as an imperative to run the "*I assume...*" entry with the rest of the list as the subelements. (The interface for the KLONE-nets-as-objects domain is based on lists like this; see pg.<associating_assertions_with_entry_s>.)

Issomorphic message elements These are the "classic" message elements: logical formulas, frames, data-base assertions. They are composite expressions that will be decomposed by their entrys into a set of "subelements" with their realization given as a mapping of those elements into the open slots of some phrase. The open phrases in the realization (as opposed to the phrase's fixed text) correspond completely to the variable sublelements of the original element—hense the name. The usual issomorphic message element is recursively constructed object with many levels—corresponding ultimately to equally many calls to entrys—all of which constitute one contiguous object from the point of view of the expert program which if printed would display all of the levels of decomposition at once, as in:

```
(frame  (duncan  (part-of (value ma (see (relation8))))
                 (ako (value (king (see (relation24)))))
                 (hq (value (dead (see (relation47))))))))
```

Here, frame is realized in terms of its single subelement (duncan ...), which in turn is realized in terms of its three subelements, and they by their own single subelements until the base of the composition is reached and we shift to arbitrary message elements like king or dead.

The requirement that an issomorphic element decompose into subelements that become part of the tree does not mean that the decomposition has to strictly follow the syntax of the expert program, only that the number of subelements is fixed (it is declared as part of the entry) and that the access routines used to find the subelements within the composite expression are also fixed. The decomposition of the (part-of ...) expression above, for example, ignores the "extra" level of list structure imposed by the token value and decomposes directly into the single element ma. Value and see are constant parts of the FRL syntax and can and should be factored out of the message enumeration process since they do not contribute any real information.

Issomorphic message elements like these properties are very common in the micro-speakers that have been experimented with. Probably because they are a very common representational style in artificial intelligence research. From the speaker's point of view they are very easy to work with since they can be taken over into standard linguistic matricies that are then specialized by default coherency rules—the speaker is able to initate reasonable quality texts without needing to

think about the details. (This is very true for the KLONE-nets-as-objects domain.) On the other hand, a dependence on issomorphic message elements will lock the speaker into a single enumeration mechanism and thereby into a single level of abstraction, i.e. the one that the designer of the expert program uses for its internal calculations. This can be awkward and can require multiple representations of the same domain information—redundancies from the expert's point of view—in order to be able to describe that information at multiple levels of abstraction. An alternative design is to employ "shopping list" message elements and have the levels of abstraction defined only within the dictionary or the speaker.

Shopping list message elements Like issomorphic elements, these elements are realized in terms of other "sub" elements. The difference is that the number of subelements is not fixed—it may differ from instance to instance—and further that the relationship between the element and its subelements may be far more arbitrary: the speaker may have brought them together only for the occasion of one speech-act. The subelements are either selected as the element is being realized by default reasoning within the element's entry (as in the entry for KL-ONE "roles", pg.<role_entry_n.y.w.>), or well before then by the actions of some earlier entry or of the speaker.

In this later case, the form of one of the element in the message is also unusual: it consists of a pairing of the element itself with a list of other message elements—the shopping list—which are the subelements from which the realization is to be constituted. The pairing is created through the application of a general purpose choice[3] with the name describe as shown below with the example from page <c303_example>.

```
(describe 'C303
          ((value-restriction C303 state)
           (next-state C205 C303)))
```

C303 is the "shopping-list message element" and the two assertions: "(value-restriction ...)") and (next-state ...) are the "shopping list" that has been choosen to describe it this time.

The "blocks" of SHRDLU's Blocks World are prime examples of shopping list message elements. Each block has a name, e.g. :B6, and that name appears in a predictable set of assertions that collectively serve to characterize it, e.g. (isa :B6 #block), (color :B6 red), (size :B6 (200 300 300)), and (#support :table :B6). A description of :B6 can either The entry for blocks (pg.<block_entry>) has one decision for each different kind of assertion that could ever apply to a block: one looks for isa assertions and knows that the third item in the assertion can be used as the sourse for a head noun, one looks for support assertions and knows that they are

3. Choices of course are actions taken by entrys, thus we are assuming that this example originated as the decision of an entry; if instead it originated with the message builder—who presumably does not share our taste in program devices—then the pairing could be brought about by something else with the same ultimate effect: an elmt-instance for the element with the describing subelements as one of its attachments.

realized as clauses and should be mapped into the qualifier slot, and so on. A specific shopping list will pick out a particular subsect of the complete set ans use them, via these decisions, to form a phrase.

Since the set of assertion-types that can describe blocks is both small and largely unique to blocks, the blocks-entry is designed as a maximal set of decisions that will be filtered by the shopping-list for each block instance. Further, since every shopping-list was created by another entry (i.e. they did not occur in messages), then a special device—meta-decisions—was used to perform the filtering and to capitalize on the fact that the reasoning was done entirely internally to the lingustic component. If, on the other hand, the shopping-lists are created earlier or the message elements in them can appear in many contexts and have well-developed entrys of their own, then an alternative form of entry is used, one that is completely general purpose, deciding only where the subelements should be positioned with respect to a common matrix and then employing the subelement's own entrys to do the actual realizing (as with C303, page <c303_entry>). Both cases are taken up at some length later in section <entrys_for_shopping_list_msg_elmts>.

Conscious versus unconscious speakers

Technical matters aside, the point of having a separable linguistics component is so that the speaker can be unconscious of the linguistic rules that are being followed as it produces a text. I mean this in a very specific sense. Even though a message that the speaker sends to the lingusitic component is couched in a domain-based, non-linguistic vocabulary, its meaning is linguistic, i.e. it is a program of instructions to the linguistic component, which, after interpretation by the dictionary, will initiate a sequence of linguistic actions. Were the speaker required to be conscious of linguistic rules, it would then be required in the construction of its messages to take into account the kinds of linguistic actions the parts of the message would initiate and the constraints imposed on those actions by linguistic rules such as "[subject] preceeds [predicate]" and "passivization effects what will be the [subject]". In saying that the speaker is allowed to be unconscious of such rules, I am saying that it is possible for a speaker to plan a message solely on the basis of non-linguistic facts about the parts of its messages; the linguistic component's understanding of the message vocabulary, of its conceptual structure and of the relations that compose it into whole messages, will be sufficient for the component to coordinate the realization of the message with the constraints of the linguistic rules without further deliberate planning by the speaker.

The idea that the speaker and linguistic component's should be separate modules—separate "planning-engines" as it were—is based on the intuition that there are different conceptual vocabularies and different plan representations involved, and that conflating them into one process is a mistake. This is enforced in the design of the linguistic component by the lack of any

provisions for monitoring of linguistic procedures, editing of the message once sent, or of scanning the linguistic plan (i.e. the surface structure tree).

Since the speaker is to plan (i.e. constraint) its message[4] solely on the basis of its own conceptual vocabulary, the fluency of its texts will be only so great as that vocabulary allows. So for example, if we were the speaker for the Macbeth domain and all we had available as our message vocabulary was FRL structures and the primitive vocabulary of the domain, then our output would be no better than, e.g.

> *Lady Macbeth persuaded Macbeth to murder Duncan in order to cause his murder. She caused his murder by persuading him to do it.*

(From the example on page <clumsy_ma>.) But if we were to add to our vocabulary notions such as focussing a text on a character or relation, or that different parts of a frame could have a different relative importance in individual rhetorical contexts, or that the large-scale order of the text made a difference in how people received it, then we could take the same "raw material" and say something like:

> *Because Lady Macbeth persuaded him to do it, Macbeth murdered Duncan.*

Of course I am not prepared, today, to present a list of the rhetorical concepts (plus their models) that should be added to every speaker, though I have written about experiments in this direction (e.g. section III.1.2, [victor]). But I believe strongly that such a list can be developed by empirical study and synthetic experiment (i.e. testing micro-speakers), and that speech-acts can be planned in terms of this "interlingua" with the speaker assured solely on the basis of its understanding of interlingua that its message-level plans can have coherent linguistic realizations.

Pitfalls of unconsciously composed messages

The speaker's concepts in the message and the linguistic concepts in the text need bear no semantic relationship to each other—the actual alorithms of the linguistic component cannot enforce a relationship and cannot even notice if there is one. Since messages are always interpreted by their dictionary, it is ultimately not the the message that matters in the selection of the text, but the entrys that the message triggers. One could design messages as trees of entry names or of abstract symbols with no counterparts elsewhere in the domain, so long as their enumeration orders "picked out" the entrys that one intended should go with that state of the domain. If its messages were this arbitrary with respect to the conceptual structure of the domain, we can say that the speaker (the message-builder rather than the dictionary) is entirely

---

4. Of the micro-speakers, logic and digitalis definately did not plan their messages, neither did the minimal text version of Macbeth although the projected version used for the example in the introduction is expected to plan. The larger-scale entrys of the KLONE-nets-as-objects domain did plan, in that they made tests and deliberated (unconsciously) over the effects that different numbers of propositions would have on the readability of the text. The tic-tac-toe domain is expected to make extensive use of plan schema to constrain the amount of detail that is included in the text according to the rhetorical situation; see [victor] for an initial discussion.

unconscious of what it is planning to say.

This ability to use "arbitrary" message elements gives the human designer a great deal of power to compensate for conceptually impoverished domains by "putting words into their mouths" via the dictionary. I used it on several occasions with the micro-speakers. In the logic domain, for example, the unitary concept of "universal quantifier" was expanded in the dictionary to discriminate number and choice, i.e. "*each*", "*every*"/"*all*", "*any*" (see pg.<conceptual_augment>). Similarly, none of the rhetorical effects in the barber proof subargument that was used to justify the universal instantiation (pg.32) were motivated from the literal text of the proof but instead were specified only in the entry of the universal instantiation inference rule. In a more sophisticated logic program, one could imagine that these two "compensations" might be unnecesary; though other kinds of compensations (such as the assignment of the name "*Giuseppi*" to the constant g) are likely to always be needed.

The indiscriminant use of arbitrary message elements to "trigger" complex phrases can be dangerous. The designer fixes the choice of phrasing to suit a particular program-audience context; if that context changes without a corresponding change in the message then the wrong information will be comunicated. The subtle "non-synonomy" of the two hedges, "*I think that...*" and "*possibly*" are an example (pg.178). Everyday computer programs that produce natural language texts through the use of templates are using a technique entirely analogous to arbitrary message elements. By fortuitous circumstances, the "conversational situations" of these programs are very simple (i.e. single sentences, no conversations, no indirect references, no contingent modifications), and consequently they do not suffer too badly from the lack of flexibility that always accompanies this technique; see [ddm_past] for an elaboration.

In the actual experience of the more complex expert programs however, completely arbitrary messages are a rare extreme. More often it is the case that those particulars of the state of the domain that the designer would want to communicate are already represented in the domain by explicit data structures that the speaker can manipulate. Furthermore, there is usually an issomorphism between the data structures and the desired English texts that a dictionary can be designed to capitalize on; in which case the easiest way to proceed is to have the message assembled directly from those structures, and to design the interface functions to link them to the entrys.

2. Technical constraints on the interface

The principal job of an interface is to associate message elements with entrys.[5] Its other jobs divide into two classes: (1) manipulating the elements for linguistic purposes, and (2) asking linguistically motivated questions about them. As discussed in section interface_functions_n.y.w., these jobs are broken down into separate functions, i.e. when an entry or a grammar-routine needs some information about a msg-elmt, it applies the appropriate interface function to the msg-elmt and receives the needed information as the value of the function. The functions that make up the interface are the only ones in the entire linguistic component that must actually "touch" objects in the domain of the speaker and expert program. For this reason, I expect that they will have to be rewritten with each new domain, or, more precisely, with each change of representational conventions and implementation language. How the functions should operate internally is left to the convenience of the designers of the various domains so long as the input/output behavior meets the specifications of the theory (pg.<interface_functions_n.y.w.>).

We will first look at the notion of a "first class object", taking it has a succinct statement of the properties that a message representation must have if an interface is to be written for it. We will close the section by considering some of the ways that have been used for form the link between message element and entry.

2.1 First class objects

When we come down to the level of a specific implementation of the linguistic component such as MUMBLE, the choice of a functionally-defined interface places one unbendable restriction on the implementation of msg-elmts, namely that they must able to be passed to and returned by a function; in the jargon of computer science, they must be *first class objects*".[6]

Abstracted away from implementation terms, the interface demands that a msg-elmt be a sign or name that can be dependably used over sucessive messages to pick out the same[7] object in the domain. The principle motivation behind this demand is the fact that the subsequent reference routine specifically looks for subsequent instances of "the same" msg-elmt as its signal to consider the use of a pronoun.

---

5. The job of decomposing the message—of identifying what the message elements *are*—is arguably prior; however, it can be left to the entrys. See section <decomposing_message_elements>.
6. Arrays are first class objects in APL, strings are in SNOBOL, only numbers are in FORTRAN, and practically everything is in LISP.
7. To be precise, the linguistic component's notion "same msg-elmt" may also be defined by on a "per-domain" basis as it is vested in the interface function same-msg-elmt. Thus giving the designer the ability to use as idiosyncratic a definition as they find necessary.

The first class object restriction is not always met. Inside there expert program certain kinds of data structures may serve their representational functions perfectly well and yet be completely unacceptable in that function once removed from the program and its implicit context. In these cases, the only recourse for the designer is to constructs an artificial set of first class objects to stand in for the original data and to use these in the messages. This was done for KL-ONE-nets-as-objects domain (pg.<artificial_first_class_objects>) when certain critical relations in the KL-ONE nets were found to be represented by unmanipulable table entries. Let us look at this problem of "underprivilaged data" as it could arise in the well known LUNAR program.

An example of a non-first-class object

The LUNAR program [lunar_short_ref][lunar_final_report] used data collected on the APOLLO-11 moon landing to answer questions about lunar geology. Its knowledge was given in the form of tables of numbers indexed by mineral and sample. The user would ask, e.g., "*What is the average plagioclase content in crystaline rocks?*", and LUNAR would reply: "*26.02778*". (Taken from [lunar_final_report] pg.G3].)

As far as LUNAR itself was concerned, "26.02778" had no meaning. It was just a number, the value of the quantified expression constructed from the parse of the geologist's question, and it existed in the program only long enough to be printed out. On the other hand, to the geologist who asked the question, 26.02778 was the extension of the concept "the average content of plagioclase in crystaline rocks". Suppose that we wanted LUNAR to be able to use this concept in its output, and proposed to do so by writing an entry based on the concept taking the number as a parameter. What would we associate this entry with? It couldn't be with the specific number 26.02778 if only because that number was the result of a calculation—it had no manifestation in LUNAR before the question was asked. Nor could it be with the data type "number", since that would tar with the same brush the individual numbers for "aluminum in breccias", for "potassium/rubidium ratios", and so on. For the same reason, it couldn't be with the variable that the number was the value of (X10, below).

The right kind of object would be one that (1) was a single expression, and (2) was sufficiently specific that it could not be mistaken by entry-for for any of the other geological concepts in LUNAR's memory. The one potential candidate is the expression from which 26.02778 was calculated, shown below.

```
(for the X10 /
        (seql (average X11 /
                        (ssunion X12 /
                                (seq volcanics) : T ;
                                (dataline (whqfile X12) X12 overall (npr* X13 / 'plag)))
                        : t))
        : t ;
        (printout X10))
```

The difficulty with using this expression is again its form: it specifies a procedure, specifically an instance of a general schema for evaluating quantified expressions. As such, it bears little direct resemblance to any text we might want to use and thus would have to be scanned to extract the parameters that determine which geological concept it is dealing with (e.g. volcanics, average, and plag). The rest of the expression would essentially be thrown away. This problem of having to filter out procedurally oriented structure before being able to make a declarative statement also came up in the work with the Digitalis Advisor.[8]

2.2 The msg-elmt to entry link

Because the dictionary entrys are so central to the operation of the linguistic component, the link from an msg-elmt to its entry is the most important of all those in the interface. It is also the one most liable to change as the representation and conventions of the domain change, hense its definition in terms of an interface function, entry-for.

All implementations of entry-for have in common an explicit association between each message element and the name of the elmt that is designated to realize it. They differ in how the association is implemented: It may be given as an immediate part of the element (i.e. as a property or as one of the fields in its record), or it may be only an association list or a discrimination net within entry-for proper. It may be direct, referring specifically to individual elements, or indirect, defining the association in terms of element *types* rather than individual elements.

Regardless of which technique is used, the link is constrained to depend only on the characteristics of the individual message element. The context of the element's position within the message or details of the state of the expert program may not influence which entry is used. (Though that context may influence what the entry decides to do once it has been selected.) The reason why there can be no reference to context is that the treatment of msg-elmts within the linguistic component will not support it: the component decomposes messages, scattering their elements to separate positions within the selected linguistic matrixes, and does not keep a record

---

8. At this point we can see another, unrelated problem that would have to be deal with if this expression were to be used as the source of a message. The geologist's vocabulary underwent some cannonicalization on route to the expression. Unless the cannonical terms can be linked back to the geologist's original expressions, we would not know which of the several English phrases that LUNAR interprets as, e.g., volcanics should be used in the reply and there would be a danger of being incoherent--giving the listening geologist the impression that we understood something other than what she said.

of the elements' former positions (because there is no independent need to).[9] With minor exceptions, the only part of the component to actually trace out the msg-elmt—entry link is the controller, and it does so only when an elmt-instance is encountered at a leaf of the tree, necessarily in issolation from the other elements. Thus as a requirement of the interface the link must be defined strictly on the same issolated basis as the message element itself is.

Contextually-defined message elements

When a message element can be identified—linked to its entry—solely on the basis of its computational properties in issolation, it will be referred to as *self-defined*. Thus far, practically every message element of the micro-speakers has met this criterion, but there have been exceptions of a predictable sort: many programs manipulate complex objects whose parts acquire a special meaning by warrant of being part of the complex without having any explicit annotation of the fact. This can happen whenever a program deals with the parts only in the context of the whole complex; the FRL "frames" (pg.<contextual_definitions>) are a good example. When one of these complexes is passed to the linguistic component and then decomposed, finding entrys for its *contextually-defined* parts can be a problem. This problem is simply that the literal manifestation of a contextually defined message element has nothing to do with its meaning. Any attempt to tie the two together just with a dictionary entry (e.g. as an "arbitrary" message element) may succeed in a few special cases, but more often than not, it will be frustrated by overgenerality (e.g. interpreting every number as a plagioclase content) or by the need for baroque excursions into the expert program in order to glean enough specific information to be certain of which entry to use.

In these cases, the correct fix is first to designate some entry(s) to be selected by whatever specificity does remain in the parts. (The dictionary for the Macbeth domain, for example, begins with just the distinction between LISP atoms and non-atoms.) This entry functions as a dispatcher, looking for a specific set of clues. The remaining part of the fix is to arrange for the placement of further clues: At the time the "parent" message element for the contextually-defined element is realized, we have a record of the meanings of its parts stored in the tree in some form. The record could be as literal as adding an attachment to the elmt-instances of each of its subelements, or as subtle as positioning a sublement in a nominal slot rather than a clausal one.

---

9. Aside from as a means of further specifying entrys, why might a record of an element's relationship to its message be useful? One possible reason (not developed here) would be to facilitate bargaining between the linguistic component and the speaker: The speaker might post several goals in its message only to have the linguistic component find that syntactic or rhetorical restrictions made it impossible to realized them all. At that point, rather than carry out its own triage policies (as done presently), the linguistic component would "ask" the speaker which of its goals it would prefer to give up. The mechanics of barganing are non-trivial, just to pose such a question there would have to be a language for describing the relation of the elements in question to the message as a whole. In the simples cases a unique index for each message element might suffice; however, reasoning about the message structure and the linguistic constraints would require structural descriptions and an abstract "meta-level" vocabulary for messages—something that would require considerable research to develop.

This technique turns out to be really no different from what happens when an entry makes a realization decision that cannot be immediately acted on and must store the decision in the tree until the controller reaches it. The only potential difference would be in the vocabulary of the records: descriptions of delayed realizations are by their nature linguistic, while the description of a message element would presumably be in terms of the concepts of the expert program.

A case in point is the "assumption" label in the transcription of a natural deduction proof as on page <barber_proof>. A formula can be called "*the assumption*" if it is the formula of the line of the proof's premis (and, of course, we are dealing with a "proof by contradiction"). Nothing about the formula itself indicates its special function, only its position within the proof. The dictionary that interprets proofs for MUMBLE realizes a line's inference rule before its formula (the formula is passed to the rule's entry as a parameter). This means that it is the entry for "premises" that must be responsible for preserving the information that its formula is "*the assumption*", which responsibility it discharges straight-forwardly by assigning the label to the formula (the details of how the label is attached are given on page <conceptual_augment>). Label assigned, the formula is now associated with an set of entrys, that will make the decision about whether to use its literal or its labeled realization.

3. Designing entrys

3.1 Basics

As has been said many times before, the basic mission of an entry is to describe how the msg-elmts linked to it by entry-for can be realized. In this section we will look at the several ways this has been done, and will look at the first steps for providing a common, transformable representation for entrys analogous to the representation for surface structure.

Conceptually, dictionary entrys have three parts:

(1) the decomposition of the msg-elmt into (instances of) its proper subelements.
(2) the set of choice-applications from which the realization is selected.
(3) the list of decision-rules that test the current context and dictate the selection to make.

Accordingly, to design an entry you must: decide how its msg-elmt (or class of msg-elmts) decomposes; decide how it could be expressed in English in terms of that decomposition; then decide how the different choices of expression vary with context. These facts and contingencies are the information the entry is to represent, and making those decisions is the bulk of the effort. The remainder of the effort goes, of course, into couching the information in the linguistic component's notation, something that is not intended to be a burden.

Decomposition

Exactly what "decomposition" means depends on the category of msg-elmt that we are dealing with. If it is an arbitrary element, its meaning for production is not explicitly represented anywhere in the domain (only in the designer's mind) and thus it does not decompose at all. If it is an issomorphic element, its structure matches that of the eventual utterance; decomposing it is a matter of identifying the "contentful" subexpressions uwithin the body of the element and extracting them from their relational matrix. The experience with issomorphic msg-elmts in the micro-speakers has been that they were always decomposed in the same way regardless of context. This made it possible to retrieve and declare the subelements all in one block at the beginning of the entry. If, on the other hand, it is a shopping-list element, its decomposition will tended to vary according to the speaker's intentions. This is because computationally, shopping-list msg-elmts are just pointers back into the domain—they have a potentially unbounded set of subelements. Since the decision of which subelements to include is intertwined with the selection of the realization, the process of retrieving the subelements and defining them within the entry is dispersed among its individual decisions.

In both the issomorphic and shopping-list cases, however, the same syntactic device is used in MUMBLE to declare ("name") the subelements and and the expressions that calculate them, namely a table of local-variable—access-expression pairs (a "variable-definition-table"; see the grammar for read-time entry expressions, page <entry_bnf>). The local-variables are just a syntactic means of referring to the subelements from within the decision-rules, and thus may have any names the designer finds memnonic. Their scope extends only as far as the decision or entry in which their variable-definition-table appears. The access-expressions that compute the subelements should use whatever functions would be appropriate for dissecting the msg-elmt in its own domain,[10] reflecting the fact that the entrys are conceptually part of the speaker. All access-expressions are evaluated by a special part of the entry-interpreter, the extra-linguistic-evaluator, which is formally part of the interface and would be rewritten as required for new expert programs.

As an illustration let us look at the entry default_property-entry from the dictionary for the Macbeth domain. As you may recall from the discussion in section winston_s_analogy_program_n.y.w., the msg-elmts of this domain are already very close to English in structure, and when in an unmarked context can be realized simply by mapping their subelements into a single-object clause. a typical msg-elmt from the Macbeth domain is

(macbeth (murder (value (duncan))))

10. When the linguistic component and the expert program do not share the same computational environment, some provision must be made to "transport" the access-expression and its calculated value between the two. Since this transport can be handled by a general mechanism in the extra-linguistic-evaluator, there is no reason why at least the names of the expert's own functions shouldn't still be used for convenience.

which is implemented as a LISP list structure just as shown. Below is the entry that performs the mapping from list structure to clause:

```
(define-entry default_property-entry (property)
    variables ((frame-name (car property))
               (property (caadr property))
               (value (caadr (cadadr property))))
    default (clause-one-object frame-name property value))
```

The variable-definition-table is indicated by the key word **variables**. It declares three local-variables: frame-name, property, and value, and defines them in terms of LISP list-structure decomposing functions that use the local-variable "property" (the entry's parameter) to reference the msg-elmt to be decomposed.

Choice-applications

The determination of how a msg-elmt should be decomposed is made hand in hand with the determination of what texts could be used to realize it. Initially, this means setting down alternative texts given just the meaning of the msg-elmt inside its domain. The next step has usually been to how these alternatives relate to the identifiable subelements of the msg-elmt—what are the "variable segments" within the texts? This leads to the identification of what in linguistic circles would be referred to as *sentential forms* for the variablized texts: schematic descriptions of the syntactic structure relating the selected words and variable segments. Using these, one picks out the choices in the established grammar that have a matching structure and parameterization, which become the basis of the choice-applications that appear in the entry. (Of course one also needs to relate the alternatives to context, which we will take up below.)

To a human, any interesting object in an expert program will suggest many alternative realizations. A toy block from Winograd's Blocks World, for example, can be described in dozens of ways just by varying the information that is included, and leaving aside variations in determiner choice and pronominalization. Here are seven: "*a block*", "*the block named superblock*", "*a red one*", "*the big block*", "*the big red block*", "*the big red block to the right of the box*", "*the block that supports one of the green cubes*". This wealth of possibilities may seem impossible to formalize; however, a few moments consideration will reveal an underlying pattern to these alternatives that one can exploit in the entry design. (As Blocks-World blocks are shopping-list style msg-elmts, I will delay discussing them until we have looked at the simpler case of issomorphic msg-elmts.)

Consider how we might realize a logical implication. Implications have two subelements, an "antecedent" and a "consequent", that will be what fill the variable segments—what we must now specify for the entry is the linguistic relationship between them. From the logician's vocabulary we get two immediate suggestions: a compound sentence based on "*if--then*", and a single-clause sentence based on the verb "*implies*". A little more thought will bring up the

conventional use of the implication as restriction on class membership (i.e. "*all men are mortal*"). From here one can expand the set of alternatives to include "transformational variations" such as exchanging the order of the if-clause and the then-clause, or to include "synonomous variations" on the lexical choices: recasting the if-clause as an embedded question, or saying "*has as a consequence that*" in place of "*implies*".

Once the alternatives have been laid out, they must be formalized—translated into the representation of the linguistic component: choices, parameters, maps, phrases, words, and constituent-structure-labels. When the designer is working with a mature grammar, there will be a library of predefined choices to select from. When working with a still growing grammar however, entry design will influence choice design, not so much in terms of the linguistic analyses used, but in the choice of parameterizations. Whether the verb "*implies*" should appear as one of the choice's parameters or as part of the phrase, for example, is largely a question of which design will be most efficient for the dictionary designer to use (also see pg.45). The formalization of the alternatives proceeds as follows: (see also the accompanying figure)

(1) The parameterization is decided upon. This determines the sentential forms.

(2) The linguistic structure that the sentential forms should have is analyzed.

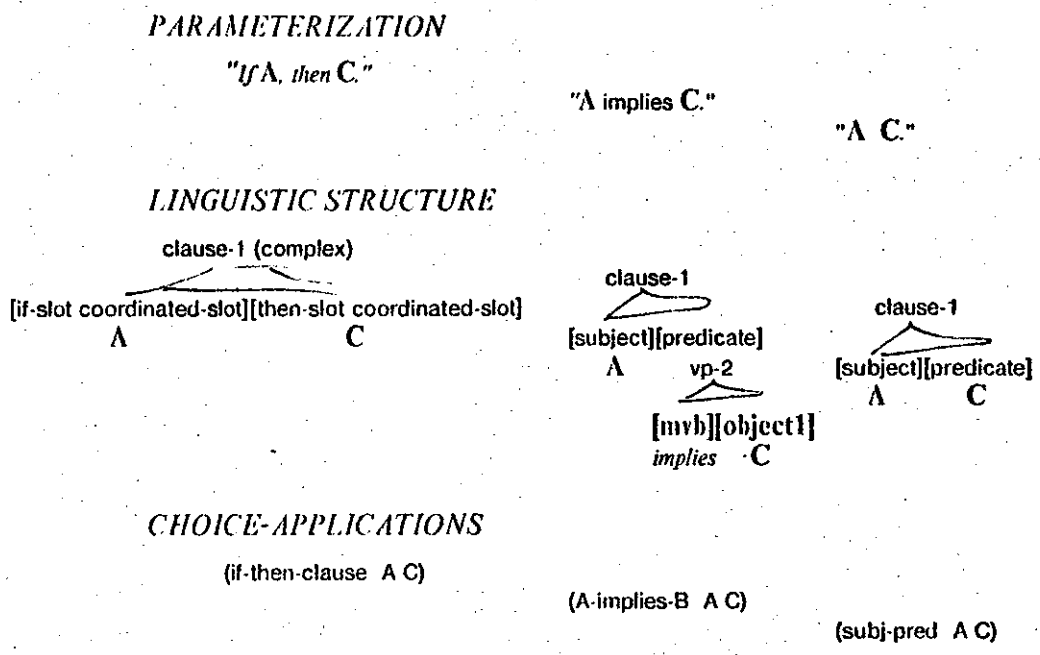
(3) The corresponding choice is found in the grammar. If it does not yet exist, it is constructed from constituent-schemas, constituent-structure-labels, and hooks as necessary.

(4) Finally, the local-variables in the entry that denote its parameters are brought together with the choice to form the choice-application.

Decision-rules

Generally speaking, it is dangerous to provide a program with alternatives without simultaniously giving it some criteria by which it can determine which to use in a given situation. Texts that appear at first glance seem to be synonomous often reveal subtle but telling differences in meaning once they are used in a certain marked context or made to interact with certain modifiers (Recall the example of "*possibly*" and "*I think that...*"; page 178.)

The experience with the micro-speakers was that alternative texts would come to mind; I would try them out in several discourse contexts, trying to imagine whether or not I would want to use them were I the speaker in that context and why; and eventually some explicit, computable, criteria would become apparent if only as an experiment. So, for example, the implication_entry was designed first to relect a stylistic ordering on the three alternatives: subject-predicate sounds better than A-implies-C, which in turn is better than if-A-then-C, second to reflect (tentative) restrictions on logical and linguistic properties that the subelements must have if a specific choice is to be usable. These criteria appear as predicates in the "load-time" representation as shown below. (Note the use of a default to insure that the decision will always make a selection.)



```
(define-entry  implication_entry  (wff)
    variables ((antec (antecedant wff))
               (conseq (consequent wff)))
    (matrix
        default  (if-then  antec conseq)
        ((A-is-a-subject-for-predicate-B  antec conseq)
         (subj-pred  antec conseq))
        ((propositionp  antec)
         (implies  antec conseq)) ))
```

It was not always possible to find usable criteria: In general, even though as a human language user one may feel that two alternatives differ in a certain way, it is (1) difficult to convert one's "feeling" into objective criteria, and (2) still harder to find within the expert program the conceptual distinctions needed to support those criteria. This comes as no surprise since it is only rarely that we are aware of making these decisions ourselves and our scientific understanding of these proximal causes for utterances is effectively nonexistent. Consequently the assignments of "contextual contingencies" to choices should be treated more as synthetic experiments—hypotheses about language use—than as facts about how people use language.

Criteria for ordering the decision-rules

In compiled entrys, the usage criteria of a decision are organized as a *discrimination net*, i.e. as a single rooted, binary branching, acyclic directed network of predicates, with choice-

applications as its terminals. To make a decision, the decision-interpreter traverses the net starting at the root, selecting branchs on the basis of the value in the present context of the predicate at the branch point; the decision is the choice-application on the particular terminal reached.

What actually must appear in a decision on the other hand is a list of decision-rules: a flatened and clumped discrimination net. The reason for not using an actual discrimination net is that tree structures are far more difficult to scan and cross-index than single-level lists. They are also more difficult to edit and display—important considerations for the designer who must actually build the entrys.

A specification of the syntax of a decision expression was given in the definitions section on page <decision>. The choice-applications and their selecting predicates are organized into decision-rules as a typographic device. Two remnants of procedural structure of the discrimination net remain: (1) the predicates are ordered within their decision-rules; and (2) the decision-rules are ordered within the decision. Since the decision-interpreter follows this ordering, moving on to the next decision-rule whenever one predicate is false and selecting the choice-application of the first decision-rule whose predicates are all true, this order must be taken seriously by the designer.

Decision-rules must be ordered within a decision so that those choice-applications that are most desirable (all other things being equal) are earlier in the list than those that are less desirable. As an additional matter of efficiency, decision-rules whose predicates are completely disjoint in the decision space (i.e. the truth or falsity of one has no implications for that of the other) should be ordered so that the most easily evaluated predicates are earliest in the order. In non-trivial decisions, there will inevitably be instances of predicates repeated across decision-rules; one either puts up with this inconvenience or uses one of the "syntactic sugaring" techniques discussed below. In any event the redundancy will go away when the entry is compiled.

As a rule of thumb, the order of predicates within a decision has been:

(1) special cases that are explicitly marked
(2) easily tested general cases
(3) elaborate or potentially time-consuming cases

Since decisions must produce values, there must be a default choice-application if the predicates of the decision-rules are not guarenteed to catch all possible cases. If for some reason a decision is truely "undefined" in some context, then the proper way to express that fact would be to use gating-condition to control the use of the decision as a whole.

We can see this order reflected in formula-entry [11] from the logic domain, shown below.

11. This entry handles both existential and universally quantified formulas: this design choice was made because it allowed the default text ("*for* <quantifier> <variable>, <body>") to be shared. Given the two quantifier-particular choices in the decision, an equally good argument could be made for their separation.

Only the matrix decision of this entry is actually spelled out here; the other three will be taken up later in the section on multiple-decision entrys.

```
(define-entry formula-entry (wff)
    variables ((body (sentence wff))
               (var (var-bound wff))
               (quantifier (qtype wff)))
    (matrix
        default (for-x-clause var body)
        ((has-attachment wff 'is-a-definition)
         (force-litteral-iff body))
        ((focus-on-variable var)
         (focussed-clause body var))
        ((equal quantifier 'universal)
         (has-attachment wff 'characterize-as-a-set)
         (focussed-clause body var))
        ((equal quantifier 'existential)
         (in-a-nominal-context-will-be-an-np body)
         (there-is-an-X body))
        ((in-a-clausal-context-will-be-a-simple-clause body)
         (clause body)))
    (express-quantification ...)
    (negation ...)
    (emphasize-polarity ...))
```

The first decision-rule ("(has-attachment...)") is used to detect and implement a decision that may already have been made by some earlier entry, namely that the formula is to be understood as a definition, e.g. "*there is space on a surface for a block if and only if that surface is the table or it has a clear top*"; it was the responsibility of the routine that choose to treat the formula as a definition to insure that it was a biconditional. The present choice acts by marking the biconditional so as to preempt the forthcoming matrix decision of its entry; see pg.<force_literal_iff_n.y.w.> for elaboration.

The second and third decision-rules also detect earlier decisions, though less specific ones: "focus" and "set-ness"; formula-entry is in effect interpreting those decisions for the specific case of a quantified formula. Note that they both selected the same choice-application.

The last two decision-rules involve potentially lengthy "simulations" of the logico-linguistic properties of the formula's body (see section reasoning_about_possible_choices_n.y.w.). That is one of the reasons for their rear-guard position, another is that the conditions they test for are not necessarily disjoint with those of the earlier decision-rules and special cases must be ordered

before general ones.

3.2 Syntactic devices to help in entry writing

Once one understands how a dictionary for a given domain should be designed, there still remains a great deal of physical labor in entering the actual entrys. This labor can be lessened by the use of some simple programming devices whose common effect is to abbreviate the amount of information that the designer must explicitly specify. The abbreviations take two forms: (1) "schema" that parameterize entire entrys, and (2) decision-extension "macros" that enable the common parts of a decision's decision-rules to be presented as such in the load-time representation but expanded out into the normal, time-efficient—space-consuming representation at run-time.

Entry schema Every micro-speaker dictionary has included dozens of "trivial" entrys, i.e. entrys that were short (e.g. just a default) and predictable (e.g. all of ten entrys might be identical except for one term). A typical example would be the entry from the Macbeth domain that associated the word "*Lady Macbeth*" with the internal token lady-macbeth. (The entry itself does little more than formalize the use of the proper name. The actual "association" is incorporated into entry-for (see pg.<contextual_definition>), causing it to select this entry in response to the token lady-macbeth.)

```
(define-entry  lady-macbeth_entry  (atom)
    (matrix
        default  (use-proper-name '|Lady Macbeth|)))
```

The Macbeth domain has one of these entrys for every token in the domain that could be named, i.e. every character, scene, location, or attribute—everything but the relations. Given this regularity, a load-time function was written to, in effect, define a schema for this "class" of entry. The function is applied to the parameter pairs, e.g.

```
(name-entry  'lady-macbeth '|Lady Macbeth|)
```

and produces their entry by instantiating the schema. It also installs the association in entry-for.

The same thing can, of course, be done for the relations as well, except that these functions/schema are necessarily more complex because of the need to extract arguments from the msg-elmts and to describe the order in which they are to appear. Arguments are coordinated by the expedient of estabishing a uniform naming convention for the local-variables that will be referred to by the entrys for, e.g., issolated FRL properties, and following that convention in naming the arguments to the function. Thus, for example, a property for which the designer wanted to use a special phrase, e.g. part-of, needs to reference the "value" field of the property

but nothing more.[12] In the expression that the designer writes for part-of (given below), the value is referenced via a local-variable of the same name. (As a further simplification, we lighten the designer's burden by allowing the expression to directly define the phrase and map, leaving the implied choice to be constructed by the function. This is appropriate in cases where the choice is used in only one entry.)

```
(define-vp-entry 'part-of
        phrase  (vp_predicate-nominative ()
                        pred-nom  (regular-np ()
                                        head "character"
                                        qualifiers  (regular-prepp  ()
                                                        prep  "in"  )))
        map  ((value . (pred-nom qualifiers prep-obj))) )
```

When the function define-vp-entry has finished it work, the result is the "full-fledged" entry below. This entry is designed by convention to realize "[predicate]'s", i.e. if the FRL property were (duncan (part-of (value (mc)))), then part-of_entry's parameter would be (part-of (value (mc))).

```
(define-entry  part-of_entry   (property-name_value)
    variables ((value   (car (cadadr property-name_value))))
    default  (part-of_choice  value))   ;this CHOICE is shown on pg.80
```

Decision-extensions The major problem with not using an actual network representation for the descrimination nets is the inability to share common predicates. To be sure, one can make up for the extra labor this implies through the liberal use of a good text editor; however, the results are not aesthetic and can be confusing to the casual (human) observer.

As a partial solution to this problem, some "syntactic sugar" has been added to the load-time design of entrys in the form of decision-extensions. Computationally, decision-extensions are subroutines (or better, macros); they make it possible to introduce a tree structure of a sort into the textual form of a decision's decision-rules, while preserving the computational efficiency of the decision once it is compiled. Syntactically, decision-extensions are independently defined entrys or decisions, i.e. lists of decision-rules, possibly with additional variable-definition-blocks. They are used in place of a choice-application inside a decision-rule of the original decision. This allows them to, in effect, distribute the predicates of the "calling" decision-rule over the new decision-rules of the "called" decision-extension.

A prime example of decision-extensions at work is afforded by the predicate-entry in the logic domain, shown in the accompanying figure. This entry consists of a matrix decision that

12. By further convention, the frame-name is always made the [subject] of the realization and consequently only a verb phrase needs to be specified.

```
(define-entry predicate-entry (wff)
    variables ((name (predicate-name wff)))
    (matrix
        ((is-slot 'clausal)
         (predicate-as-clause wff))
        ((is-slot 'predicate)
         (predicate-as-clause wff))      ;EQUI-NP-DELETION applies obligatorily
        ((is-slot 'nominal)
         (predicate-as-np wff))                                    ;
        ((is-slot 'modifiers)
         (predicate-as-np-mod wff))
        ((is-slot 'head)
         (head_gets_name name current-np)) ))

(define-decision predicate-as-clause (wff)
    variables ((arguments (arguments-of wff)))
    ((will-be name 'clause)  ;e.g. "shaves"
     (return-application-of-entry-to-args name (entry-for name) arguments))
    ((will-be name 'np) ;e.g. "barber"
     (clause-predicate-nominative name))
    ((will-be name '(adjp adjective classifier))
     (clause-predicate-adjective name)))

(define-decision predicate-as-np (wff)
    variables ((var (single-argument-to wff)))
    ((will-be name 'np)
     (variable-described-as-name var name)))

(define-decision predicate-as-mod (wff)
    variables ((var (single-argument-to wff)))
    default (return-application-of-entry-to-args name var))
```

---

does a dispatch on the features of the current-slot, accompanied by three independent decisions acting as decision-extensions that handle all the decision-making that depends on properties of logical expressions.

4. Multi-decision

Making one decision means selecting one choice. Allowing entrys to have multiple decisions is thus a way to enable the single "choice" of an entry to be constructed by the combined effects of many, general-purpose choices, rather than by requiring the creation of a single, special-purpose choice which would have the same effects but which would not serve more than its own entry. Without multi-decision entrys, it would be impossible to put a bound on the number of special-purpose choices that would be needed since (presumably) it would grow with the number of entrys in the dictionary. On the other hand, the use of general-purpose choices, combined as needed through the device of multi-decision entrys, holds out the strong possibility of arriving at a fixed choices vocabulary.

4.1 Factoring out common actions

As an example of modularity in entry design consider the express-quantification in formula-entry (pg.207). The matrix decision of that entry had five possible choices for the specification of its matrix, i.e. for the specification of the linguistic structure that replaces the formula in the tree as its realization. Regardless of which choice was selected, the variable instance bound by the formula was to be marked by an attachment that would insure that when it was finally realized, the choice of determiner would be based on the quantifier of the formula. Rather than create five new choices consisting of the originals plus the determiner-fixing action, I instead added a second decision to the entry which applied regardless of what the matrix decision had been.

```
(define-entry formula-entry (wff)
    variables ((body (sentence wff))
               (var (var-bound wff))
               (quantifier (qtype wff)))
    (matrix... )
    (express-quantification
        default (mark-as-quantified var quantifier))
    ...)

(define-choice mark-as-quantified (object type)
        actions ((attach-to object 'quantified-by type)))
```

4.2 Overriding default decisions

Default-decisions are a means of dealing with locally unpredictable relations such as negation or intensification, or of supplying default information not always given in the message such as determiners or tense; see sections <default_decisions> and <attachments_and_default_decisions>. default-decisions are "owned" directly by the entry-interpreter rather than individual entrys and they apply after all of the decisions of the entry's have been completed. That they are "defaults" means that they may be overriden in specific cases. This is done simply by including a decision of the same decision-name with the entry.

One of MUMBLE's default-decisions emphasizes the polarity of a clause when signaled to do so by an attachment on the elmt-instance being realized (e.g. "*I.ady Macbeth did NOT kill anyone*"). The choices of this decision (i.e. emphatic "*do*" and stressed "*not*") will not always be the best ones for an entry's constructions however, at which times the default should be overridden. The final two decisions of the formula-entry are a good example: that entry sometimes uses an existential "*there*" construction, and when that happens, negation is best emphasized by forcing the determiner of the direct object to be "*no*", e.g. "*there is no such barber*".

```
(define-entry formula-entry (wff)
    (matrix... )
    (emphasize-polarity
        gating-condition (and (has-attachment current-instance 'emphasize-polarity)
                              (has-attachment current-instance 'negated)
                              (matrix-choice-was 'there-is-an-X))
        default (negate-variable var))
    (negation
        gating-condition (decision-was-made 'emphasize-polarity)
        default (do-nothing))  ;i.e. the DECISION was already made by "emphasize-polarity"
```

4.3 Contingent Decisions

The two decisions above are intimately connected: the second is made if and only if the first was, and the first is itself contingent on the choice selected by the matrix. That connection is there to avoid making a decision twice; in the example below, two decisions are connected in order to continue what is effectively a multi-decision choice, i.e. having made one decision, the speaker is now forced to make another as well.

```
(define-entry iff-entry (wff)
    variables ((antec (antecedant wff))
               (conseq (consequent wff))
               (merge-var (merge-able wff))) ;described on pg.185
    (matrix
        default (literal-iff antec conseq)
        ((is-region 'formal)
         (literal-if antec conseq))
        ((merge-able wff)
         (not (equal merge-var 'constant))
         (iff-restricted-relative antec
                                  conseq
                                  (variable-to-merge-on wff))))
    (realize_the_only-if
        gating-condition (matrix-choice-was 'iff-restricted-relative)
        default (synonym-set '((all-and-only merge-var)
                               (and-none-else merge-var))))
```

When you design in terms of multiple decisions, you invariably need some kind of "inter-decisional control"—some way of making the use of a decision contingent upon the context. The technique illustrated in the three decisions above makes a decision rather like a decision-rule, i.e. a set of predicates (the gating-condition) is used to define the preconditions under which the decision can be made.

While the technique is adequate to the job, its dependence on the execution order of the decisions in the entry and its unconstrained power (i.e. any predicate can be used in a gating-condition) mean that the control paths it defines are liable to be obscure (i.e. difficult for another designer to follow in the text of the entry) and that it will not structurally distinguish control regimens that are principled from those that are ad-hoc.

On the other hand, unless the semantic distribution of the decisions—the semantic, syntactic, or rhetorical contributions of the individual decisions to the entry as a whole—are themselves principled, it will be impossible to design a rational improvement to the gating-condition technique. A candidate rational improvement will be discussed momentarily.

4.4 Domain centered decisions

A very natural way to organize an entry is in terms of the domain facts or attributes that are to be included in the realization: one fact — one decision. One starts with the class of objects to be described, say "blocks" in the Blocks World, and proceeds to list the attributes by which they can be described, e.g. their color, their size, their location, their support relationships, and the fact that they are a "block" rather than a "pyramid" or a "ball". Each attribute is established in its own decision, with a variable-definition-block to extract from the domain the value of the attribute for an individual instances of blocks, and matrix decision by which to determine how to express the value with respect to a matrix construction—a noun phrase in the case of blocks. The

result is shown below.[13]

```
(define-entry blocks-entry (block)
    (matrix
        default (regular-noun-phrase))
    (block-ness
        default (head_gets block matrix))
    (color
        variables ((color-name *dummy*)
                   (assertion !(color ,block >color-name)))
        default (modifiers_gets color-name matrix))
    (size
        variables ((size (quantize-size !(size ,block >dimensions))))
        default (modifiers_gets size matrix))
    (location
        variables ((loc (describe-location !(size ,block >dimensions))))
        default (qualifiers_gets loc matrix))
    (support-relation
        variables ((relation (or !(supports ,block >other-block)
                                 !(supports >other-block ,block))))
        default (qualifiers_gets relation matrix))
    (determiner
        default (determiner_gets the)))
```

When this entry is interpreted, the matrix is constructed first—empty of any constituents. When the other decisions are interpreted, each either fills one of the slots defined by the matrix or adds to the constituents already there (assuming those slots are designed with hooks and grammar-routines so as to cache the constituents (e.g. multiple adjectives) as they are recieved).

As written here, the blocks-entry always produces maximal realizations (e.g "*The big red block that is at the left-rear of the table and that supports* <B3>"), and has omitted several ways of describing blocks that were mututally incompatable with this "main sequence". We need some means of controlling which decisions are used, perhaps grouping them into families. This would be very awkward to do just with gating-conditions, especially as soon as we try to factor in discourse predicates that would, say, cause attributes to be omitted if the audience already knew them. "Meta-decisions" will be introduced later as a just such an alternative controlling device.

4.5 Grammatically annotated decisions

When decisions are annotated in grammatical terms, it is possible to manipulate them using rules that are triggered by the grammatical annotation alone, without requiring any knowledge of their conceptual content. In the section on the "interlingua", we will look at grammar-guided

13. The "!", ",", and ">" characters in the access-expressions are a notation for pattern matching: variables prefixed with ">" are instantiated by the pattern matching operation, and variables prefixed with "," have their values employed as literals just like all the non-variables in the expression. The matching operation returns the set of matching assertions that it finds and binds any instantiated variables as a side-effect, hense the "*dummy*" binding in the variable-definition-blocks of the color decision.

manipulation that is associated with subsequent reference and uses meta-decisions. Here we will look at one that is special to the design of the dictionary of the logic domain (though it would be applicable in analogous situations).

In the logic domain, variables are introduced into the tree either in issolation ("*Everything that is a man is mortal*") or as the argument of some categorical predicate ("*All men are mortal*"). In both cases, the reference that is entered into the discourse history for the benefit of later pronominalization is to the variable, never the predication; instead, the predicate is interpreted as a way to describe the variable, and entered into the history as a modifier. This interpretation is done very literally by taking the very decision that the predicate itself employs in issolation (e.g. "*There is a man.*") and using it to replace the equivalent decision of the variable's.

Below are the "normal" entrys for variables and for the predicate man. Note that the decision-names used are identical to the slot-names affected by their respective decisions.

```
(define-entry variable_entry (var)
    (matrix default (regular-noun-phrase))
    (head
        default (head_gets thing matrix)
        ((humanp var)
         (head_gets person matrix)) )
    (determiner
        default (mark-as-quantified var quantifier)))  ;this is a decision-extension

(define-entry man_entry ()
    (matrix default (regular-noun-phrase))
    (head default (head_gets_classname 'man matrix)))
    ;the determiner is provided by grammatical defaults
```

When realizing a predicatation like man(x) as a noun phrase (pg.209), the choice variable-described-as-name will be selected. This choice takes the elmt-instance of the variable and "edits" its decisions-to-make property so as to substitute the head decision of the entry for the predicate (man_entry) for the original head decision of the variable_entry.

```
(define-choice variable-described-as-name (var name)
    element-returned var
    actions ((substitute-the-decision (get-decision 'head name)
                                      for 'head
                                      in var)))
```

As a result of this action, that particular instance of the variable will now be realized using the head decision of an entry about which it had no *a priori* knowledge—an event that is possible only because the term "head" was used conventionally in that dictionary, thereby assuring that there would be no unwanted interaction between the new decision and the remaining originals in the variable-entry. entry conventions like this one are the key to the development of entry-

manipulating rules such as used for non-pronominal subsequent reference.

5. Entrys for shopping-list msg-elmts

All of the entrys looked at thus far have either been for issomorphic msg-elmts or shopping-list msg-elmts whose subelements were selected on the basis of defaults embedded in their entrys (i.e. their shopping lists were "implicit"). Another alternative is a shopping-list msg-elmt where the decision as to what properties should be used to describe the element is made by the speaker or by an earlier choice before the element's own entry is ever reached. The decision is supplied to the entry as a list of subelements—an "explicit" shopping list. For example, earlier on page <describe_c205> a hypothetical entry decided to have the KLONE concept C205 described in terms of three facts about it, using the general purpose choice describe.

```
(Describe 'C205
          '((specializes C205 JUMP-ARC)[14]   ;source of "jump arc"
            (type C205 generic)               ;source of "the ...s"
            (source-state C205 C303)))        ;source of "...from S/NP"
```

5.1 Interpreting shopping lists

When a msg-elmt is used as part of a shopping list, its meaning (to the linguistic component) may be only distantly related to its meaning when it is used in issolation. For example, (specializes C205 JUMP-ARC) by itself will become the clause "*"C205" is a kind of jump arc*", but when part of a list of facts selected to describe C205, its meaning is instead the contribution of the generic name "*jump arc*" to a noun phrase.

While one can imagine designing a manipulation of the entry for specializes that might be able to extract a usable "(say *"jump arc"*)" decision analogous to the earlier case where predicate-entry was canibalized to describe logical variables, it is unlikely that this could be done for all of the msg-elmt classes that might appear in a shopping list. Instead, it seems that the entrys of message elements that could ever be part of a shopping list should be designed from the start with the need to support that role in mind. Two different techniques for designing this support have been explored in the micro-speakers. Both will be discussed after the common part of the procedure has been described.

---

14. The possibility of structuring lists of "kernal propositions" such as this by using boolean functions or discourse relations (as in the example in the introduction) then defining transformations in terms of that common relational vocabulary is being developed as part of the next step in this research.

Multiplying a common decision in the elmt-instance From a technical point of view, the most striking fact about a shopping list description of a message element is that the number of describing facts can vary arbitrarily. It implies that an entirely new technique for associating decisions with entrys must be used in order to accomadate this arbitrariness. As we will see, the technique adopted is essentially the same as will be used for meta-decisions.

Within the realization procedure, the entry-interpreter uses the elmt-instance's property decisions-to-make to determine what decisions it should use and in what order. This property is usually a copy of the one on the elmt-instance's entry, however, it need not be: it can be set to an arbitrary list of decisions at any time before the elmt-instance's realization is begun. The unmarked choice for defining a shopping list, describe, uses this facility to insure that arbitrary numbers of facts can be accomadated. Describe takes two parameters, the msg-elmt to be described—the one that the resulting linguistic structure will denote—and the list of msg-elmts that are to be its description. Its body consists of one purpose action that builds a "custom" elmt-instance whose real-msg-elmt is, e.g., C205, whose entry-for is shopping-list-NP-entry,[15] whose entry-arguments-for is the shopping list—the list of msg-elmts being picked as the description, and whose decisions-to-make is a list consisting of as many copies of the common decision (below) as there are msg-elmts in the shopping list.

```
inst-1
    real-msg-elmt  C205
    entry-for  shopping-list-NP-entry
    entry-arguments-for  ((specializes C205 JUMP-ARC)
                          (type C205 generic)
                          (sourse-state C205 C303))
    decisions-to-make '(fancy-shopper fancy-shopper fancy-shopper fancy-shopper)

(define-entry  shopping-list-NP-entry  (the-shopping-list)
    variables  ((shopping-list  the-shopping-list))
    (matrix
        default  (regular-np)))
```

An unmarked, itterable decision for a shopping list Exactly what the itterated decision consists of depends on the outcome of the design decision as to how the entrys for the msg-elmts on a shopping list are to contribute to the realization process. One possibility is that each message element on the list can be adequately realized as one word or phrase. This would allow the simplest possible itterated decision, call it ("simple-shopper"), since all it would do would be to position each element in the matrix, leaving its realization to its own entry. The only work that would have to be done would be deciding which element went into which slot, and this can be done on the basis of their entrys' will-be properties.

---

15. We know that the realization must be a noun phrase because of its context in the tree.

The tic-tac-toe domain can use simple-shopper. There the phrase "*the corner below mine and opposite yours*", for example, comes from the shopping list:

```
(describe 'square-9
          '((corner square-9)
            (below square-3 square-9)
            (opposite square-1 square-9)))
```

via the constituent structure:

```
                        np-1

[determiner]   [head]    [qualifier]__
                  (corner square-9)
                                    |multiple-contents ((below square-3 square-9)
                                                        (opposite square-1 square-9))
```

---------- patch ----------
which is built by iteration of this simple-shopper @@

In this technique of "positioning intact", entrys must be designed so as to notice when their elmt-instances are the contents of, e.g., the [head] of a noun phrase, rather that, e.g, in a slot marked clausal. The predicate-entry on page 209 is an example of this.

If there is no reason why a given elmt-instance should find itself in this kind of embedded context other than because it is part of a shopping list, then perhaps the proper context to be noticed isn't a slot-name but the fact of being part of the shopping list. This is especially true when the element's meaning is different when it is functioning as part of a shopping list than when it appears by itself, at least to judge from my initial experience's with the real KLONE domain.

In such cases, there is little reason to avoid actually realizing these elmt-instances during the realization of the shopping list. That is, rather than have an intermediate stage where the items on the list occupy slots in the linguistic matrix, we go directly from the positioning decision to their realization decisions, and fill the slots with words and phrases. In any case, this alternate technique of "positioning by realizing" *must* be used when an element cannot be realized as a unitary object but instead effects several aspects of the matrix at once, such as (type C205 generic) above which is responsible for both determiner and number.

The same archetecture of a repeated common decision would still be used, with the addition that the "fancy-shopper" decision must be empowered to call the realization procedure recursively within the entry where it is operating. This constitutes a non-trivial extension of the theory given in chapter two, but if suitably constrained, could be very powerful. It may, for example, permit one to write a dictionary for the generic concepts of a semantic net—inheritance

hierachy—in such a way that entrys for individual instances can be compiled automatically from the network without human intervention. On the other hand, this style of operation is very new in this research and must be considerably shaken-down and developed before it can be reported on in any detail

6. The beginnings of an Interlingua

An *interlingua* is a language used "between", in this case between the speaker and the linguistic component: the linguistic component speaks a language of grammatical rules and linguistic actions; the speakers use the language of their expert programs, whatever that may be. It is because the languages—representations and conceptualizations—of the expert programs will inevitably be different from each other that the design decision was made not to develop a general interlingua for messages but instead to use the data structures of the different programs directly.

If the message languages of individual domains vary this much (the whole system of interface functions, for example, is predicated on the inevitability of possibly drastic differences), what is the point of an entire section on "the beginnings of an interlingua"? One general reason is that in the future there need not be differences: If the designers of new expert programs have natural language processing in mind, they may be able to shift their free design decisions so as to make production easier, given, of course, an adequate description of what "easier" would amount to; this is discussed with respect to specific representations in appendix VII.B.1.3.

Without waiting for future programs, we can look at two sources of specific phenomena that may lead to the aggregation of an interlingua from the bottom as it were. The first of these is linguistic concepts that have no counter-parts in the expert programs, and the second is "abstract" linguistic structures created via subroutines and common labels within the dictionary. Both of these depend (as does any language) on strict adherance to convention, and we will see in this section what some of those conventions could be.

6.1 Concepts unique to language

The fact that this research has been about programs rather than people may have introduced "unnatural" problems as a result of the relative conceptual impoverishment of the programs. When working with programs, one must "fortify" their messages with additional assumptions and defaults in order to support decisions required to fully specify natural language texts. People, one believes, already have the needed concepts of discourse and linguistic communication at hand when they decide what to say and thus go to their "lingustic components"

with messages fully specified.[16]

In order to make up for the deficiencies of individual programs, the linguistic component includes a number of decisions associated by default with linguistic events such as determiner selection, the calculation of agreement, or the selection (or omission) of a relative pronoun—events which one does not expect your garden-variety expert program to care about let alone know how to decide. If the identical decisions could be used from expert to expert, they would effectively constitute an interlingua. There are enough hooks[17] in the design of the linguistic component to allow a designer to create entrys with choices so thorough that a message could specify every linguistic detail of the text down to the word order and morphology. (Perhaps these would be useful to a poet.) The fact that such choices would have to preempt a great many grammar-decisions and constituent-schemas demonstrates that the existing grammar already has preconceptions about the level of decision-making that its speakers will choose to make via their dictionaries.

A uniformity between speakers could come about for either of two reasons: either the test criteria of the decisions might be entirely linguistic and thus insulated from the speakers' different representations, or they might be tested via a common set of predicates, effectively expanding the set of interface functions that must be defined with each new speaker/expert-program combination.

There were not many attempts to develop an actual interlingua among the existing micro-speakers. This was more because of their sequential development and differing linguistic contents, than because of intrinsic difficulties. The cases below are practically the entire repetoire; they should, however, be taken as examples of what can be done, and not as a maximal list of what is possible.

Determiners for predicate nominatives In texts like "*Socrates is a man*" or "*Duncan is a character in "Macbeth"*", the predicate noun phrase is cast by the language into the role of a description or label. This means that regardless of its ontological position in their expert programs, the message source for "*man*" or "*character*" is interpreted here as though it was the name of a generic concept. As this interpretation was forced by linguistic context, that same circumstance should be able to insure that the noun phrases have the correct form for generics, i.e. have the correct determiners.

---

16. Of course it is only an assumption that people have the conceptual basis to support the distinctions they make in language. Consider that if evidence of these concepts should appear nowhere else in human behavior except in language use, then given the presently impoverished state of neuropsychology we can not prove that those concepts are not solely the products of linguistic processing.

17. The word "hook" is used here in a technical sense developed by Erik Sandewall and myself to describe provisions in a program's design that allow the smooth integration of ideosyncratic specializing information into a common control structure. Specific programming constructs and disciplines are discussed in [sandewall_hooks].

A default "determiner" decision was added to the grammar to cover this case. it can be written as:

```
(define-decision default-determiner (np-matrix)
    gating-condition ((is-slot 'pred-nom))
    ((elmt-pluralp current-subject)
     (attach-to np-matrix 'plural t))
    ((not (elmt-pluralp current-subject))
     (determiner-gets "a")))
```

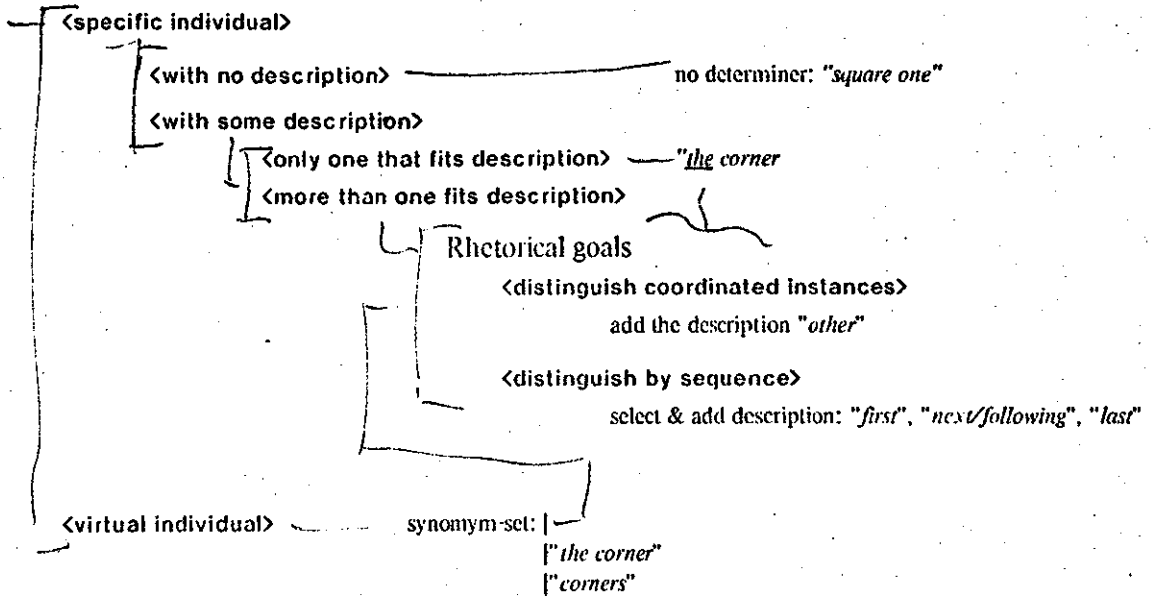
This decision was one of the default-decisions associated with noun phrases (n.b. proper names were not analysed as noun phrases), and thus applied in every case where a noun phrase was produced as the contents of a [pred-nom] slot and the entry involved had no determiner decision of its own. In the two domains where it was used: logic and Macbeth, there was no notion of a generic/instance distinction except for this linguistic one, and this default was the sole source of determiners.

In a domain that did have its own generic/instance concept (e.g. KL-ONE), this default would be superfluous as written—there would be no need to fortify the domain in this way—and would be replaced by a default at a higher level, e.g. having the speaker build its messages in such a way that predicate nominative constructions were only made with generic message elements. If that could not be guarenteed—if it were possible that non-generic or "neutral" elements might be used at some times—it would be very difficult to reinstate the linguistic default since it would now have to be coordinated with already existing decision-rules in the element's own determiner decision, (i.e. that that discriminated that neutrals from the generics) and that is very difficult to do automatically.

Uniform criteria for determiner selection The beginings of an example of conventions enforced by augmenting the interface functions appears in the tic-tac-toe domain. Here the goal is to define a general purpose determiner decision—one that would apply to every referring object in the domain without the need for special cases. The strategim adopted is to design a set of predicates—concepts really—that will define the system of choices that are involved in the proper use of a determiner; special cases are then to be "pushed off" to the definitions of these predicates. The enterprise is still underway, i.e. the existing "determiner system" is adequate to the present needs of the tic-tac-toe domain, but obviously inadequate as a formalization of the criteria behind determiner choices given the full range of choices permitted by English. The system is shown below.

In the ontology of the tic-tac-toe domain, the only relevant object types are specific individuals (e.g. the squares, or the diagonals), virtual individuals (i.e. the variables in rules--ultimately bound to specific individuals), and generic labels (i.e. descriptions applied to individuals). One would like, but usually cannot arrange, for these distinctions to be carried over

into the determiner system: English has its own ideas about what is important.



The examples above have been cases where the demands of natural language forced conceptual distinctions that were present in a given domain. These lead to an interlingua of sorts based on a common set of grammar-supplied defaults to handle the distinctions, and a set of common conventions in their triggering—a discipline in message construction—to insure that the intentional contexts leading to these defaults would in fact be the ones expected when they were designed.

Another way in which linguistic phenomena will prompt an interlingua is by supplying a richer conceptal vocabulary for message construction. These would be concepts involved in communication via natural language rather than the concepts of linguistic form which are, of course, left to the linguistic component. For example, we can look back to the example message from the Macbeth domain and see the concepts "sequence", "time-frame", and "focus", used by the (projected) speaker in its messages even though they are not native to the Analogy program's own conceptual base. More subtlely, natural languages provide compact, powerful pronominal expressions like "*each other*" or "*respectively*" which permit the succinct expression of some otherwise complex relationships and thereby could encourage their use in a message.

To be properly part of an interlingua, concepts like these would have to have a common meaning across speakers: in grouping a set of propositions as, say, a "sequence", each speaker would have to intend that the linguistic component understand the "sequence relation" in the

same way: construct the same linguistic matrix, consult the same decisions. This "meaning" can be subtle: What defaults are left to the linguistic component when it sees "sequence" in a message? Can it make its own decisions as to how propositions will be clumped into sentences? Will its criteria for sentencehood be the same from speaker to speaker? What kinds of reduction or merging of common seqments are allowed?

At this point, I am not in a position to suggest how suggest how a concept like "sequence" should be formalized since the emphasis of the present research has been with the linguistic component and not with speakers. It has elaborated what such a formalism must address—a necessary first step—but it can make no specific offerings until several extensive, independently principled speakers have been investigated.

6.2 Conventions

With no exceptions, every choice discussed in this paper specifies some literal aspect of extended surface structure: a pattern of specific auxiliaries rather than "the narrative past" or "the hypothetical future"; the words "*the*" or "*a*" rather than "specific" or "generic" determiners. This "concretism" is a way to avoid undo generalization of the message element-linguistic construct links: A concept like the hypothetical future is an abstraction whose behavior and constitution linguistics may disagree about; "*could be being*", on the other hand, is an indisputable English auxiliary phrase about whose use in a given textual context one can have a definate stylistic or grammatical opinion. By maintaining this kind of theoretical conservatism, one can be sure that the theories of language use (i.e. dictionaries) that one develops, while surely too small, are at least not houses of cards.

But having begun conservatively, we should nevertheless take advantage of generalizations when they arise. In particular, when sucessive micro-speakers are found to make the same choices in response to the same circumstances, we should take steps to capture this regularity in some kind of structure and rule combination. We can, on this basis, build up from the theoretically certain but conceptually impoverished level of extended surface structure to a level which, while more abstract and questionable ontologically, would be closer to the level at which the micro-speakers actually reason and thus be more powerful.

In practical terms, what this amounts to is an extension of the grammar—the conventional knowledge about production common to all speakers. A not implausible example might be for every speaker to share the entrys for locial AND or for predication. For this to be possible, the micro-speakers would have to first share conventions about how, e.g., a predication was represented: how its arguments were accessed, how the name of the predicate was associated with an entry; then they would have to agree on what a predication meant: how it could be interpreted in terms of linguistic constructions, and what contextual contingencies were relevant—for example the description/predication paradigm would have to be shared (e.g. man(x) is realizable

as both "*a man*" and "*X is a man*"). If these conventions, these consistencies in notation and intention, were sucessfully propagated among the micro-speakers, then they could be left as details for the rule system, just as at the present time the details of what it means to use a clause or how one signals number agreement in a verb is left to the rules of English grammar. They would no longer be points of difference whose details had to be specified with each new speaker.

Entrys manipulating other entrys

One of the tenets of this theory of language production is incremental processing: each decision-maker should be responsible for no more that it can justify on the basis on what it knows; as a corollary, each decision-maker should also make as large a decision as it knows how to. The key, of course, is the specification of what an entry can "know". The natural limitation on the application of a general rule (e.g. a transformation) during the realization procedure is the subelements of the msg-elmt being realized—one level of breadth only (plus the limitations on access to grammatical context dictated by the controller); but what should the limits on the "statis" knowledge that one entry can have of the other parts of the dictionary? how much should an entry know about the criteria and potential decisions of other entrys in the dictionary? In particular, should it be able to know that the entry for one of its subelements is able to make a particular choice?

A case in point is formula-entry (pg.207): its matrix decision includes a choice-application that preempts the later matrix choice of the entry for biconditionals 212), forcing it to be literal-iff. Now this action makes no sense unless that entry in fact has literal-iff as one of its normal possibilities; indeed, since MUMBLE implements the preemption without any checking, a "miss-judgement" by formula-entry would cause a fatal error. Still, formula-entry does not first check that its action is legitimate; it effectively presumes its legitimacy, based on the omniscience of its designer.

This kind of handcrafted dependency between entrys is dangerous in the long haul: (1) because it is prone to human error; and (2) because it is *ad-hoc*: there are no systematic, grammar-based constraints on what parts of one entry may be known to others—no reason to believe that the connections would have any generality. Communication between entrys via attachments is an improvement, especially when the mark may be placed by several different entrys and leaves open exactly how the instruction is to be implemented in a given case (as was true with the case of marking a proposition for "emphasized polarity", pg.<emphasize_polarity>).

In the case above, the preemption was made in response to an attachment on the formula indicating that it was intended to be "a definition". Formula-entry was the first place in the enumeration of the message where the attachment would be noticed, making it the place where it had to be acted on. If the dictionary for this logic domain had not been strictly based on predicate calculus operators and inference rules, the better thing to have done would have been to define an

internal "type" to represent definitions and then to give it an entry of its own. As it stands, we have the effect of a separate entry by means of this *ad-hoc* attachment and preemption spanning two "regular" entrys.

meta-decisions

Besides these *ad-hoc* cases, there are many times in the designing of a dictionary that the most general way to express some linguistic phenomena is as a filtering or editing operation on the decisions of existing entrys. If the phenomena is to be incorporated into the grammar of the linguistic component, e.g. as part of the subsequent reference heuristics, then this ability is a requirement. The device I have settled on for this I call a *meta-DECISION*.

A meta-decision is no different syntactically or functionally from any other decision; the difference lies in what its choices do. You may recall that what decisions a msg-elmt's entry will make, as well as their order, is actually determined by the decisions-to-make property of the msg-elmt's individual elmt-instances. Usually this property is a copy of the property of the same name on the entry; however, part of the specialization of an elmt-instance can be the specialization of its decisions-to-make property: some decisions might be left off, others replaced by different versions, or new ones added. Using this technique, one can write choices that do this kind of "editing" as part of their marking-actions, returning elmt-instances whose entry's will perform very differently from the normal case. (Recall the use of this technique with logical variables described by predicates (pg.<variable_entry_n.y.w.>).)

For a quick example of how a meta-entry is used (there will be longer ones in the discussion of subsequent reference) we can look back to the blocks-entry of page <blocks_entry>. That entry has one decision for each different property a block could have in Winograd's Blocks World, i.e. being a "block", having a color, a size, or a location, and a support relation.[18] To use a meta-decision, we change entry-for to link specific blocks (e.g. B6, B3) to a new entry, call it block-strategy-entry. Block-strategy-entry is given an instance of, e.g. B6 as its parameter and returns the same instance as its realization except that now the elmt-instance's entry-for property has been changed to blocks-entry and its decisions-to-make property is some specifically selected set of decisions.

Among the choices one could write for blocks-strategy-entry would be one that picked out the matrix, plus block-ness, plus color, plus size, plus determiner to produce (after blocks-entry was finished): "*the big red block*", or one that picked just the "having a name" decision to produce "*superblock*". With some computation to determine the state of the domain, one could duplicate Winograd's original style-criterion, namely mentioning only so many of a block's properties as are

18. Now that there is a facility for using only subsets of the entry's decisions, one can add decisions for "having a name" and "sequence within a related set of blocks" as in "*one of the green cubes*" or "*another of the blocks in the box*".

necessary to disambiguate it from all of the other blocks in the domain given an a priori ordering of preferences. The computation would yield a list of decisions, one for each property needed, which would be installed as the decisions-to-make of the elmt-instance being realized, thus "filtering" the blocks-entry so that it uses the minimum of its decisions. After elaborating some of blocks-entry's decisions, particularly the determiner decision, one could even define generic "subconcepts" solely within the dictionary. That is, given an individual from the blocks world, one could construct generic abstractions of it, e.g. "*a small block*", "*a block that supports another block*", solely on the basis of earlier decisions by the speaker.

One "problem" with meta-decisions is that the computations one is typically interested in, such as the computation above of the minimum number of properties needed to disambiguate some object, is very awkward to couch in terms of decision-tree selecting from a fixed set of possible choices. The space of choices is very large, possibly the powerset of the entry's list of decisions, and the decision-trees equally so, since they would require explicitly representing every state in the FSM model of the decision algorithm. What I have found myself doing to avoid this is clumping choices together, e.g. having a single choice, say one named minimum-disambiguation-strategy, and then "burying" the computations needed to select the precise list of decisions into the actions of this choice. This turns out to be not too missguided a technique since, in developing heuristics to reason about past decisions, I find that one is only interested in choices at just such an abstract, "strategic" level, with the details of precisely which pattern was selected an why not being important.

The idea of operating on an entry at a meta-level that can resolve individual decisions but not their contents is an attempt to answer, in a disciplined way, the question of how much of the internal contingencies and options of an entry should be known by either a grammar-routine or by another entry. It seems to me to be possible to develop a closed theoretical vocabulary for annotating decisions: one can relate them to syntactic or rhetorical categories they create for example, or to generalizations about the kinds of subelements they expect. On the other hand, it seems extremely unlikely that a comparable taxonomy could be devised for predicates and decision-trees, but without any coherent description of a decision's "substructure", it will be impossible to develop a rule-governed, automatic means of manipulating or reasoning about them.

I have not yet developed a vocabulary for decisions, indeed, many of the ideas in this section were not even thought of until this document was being written. However, I believe that the "experiments" conducted thus far with individual entrys in individual micro-speakers have shown the general outline that the vocabulary will take. In particular, the fact that the experiments have centered around the statement of general rules for subsequent reference—probably the place where the linguistic grammar extends most deeply into the pragmatic domain of the speaker—should be evidence that the techniques will be extendable.

7. Non-pronominal subsequent reference

7.1 Alternatives to pronouns

In order for a text to be coherent, it must treat subsequent references to an object differently from initial ones. This is because a natural language text is expected to reflect the fact that its audience has assimilated what it has read, remembers it, and need not be told it again and again. This is not a logical requirement—programming languages, for example, do not follow it—rather it is a convention, part of a larger trade-off that dictates that the descriptive devices of a natural language are to be used as an intentionally controlled focusing mechanism: like the camera in a well-directed movie, language highlights what the speaker thinks is new and important and diminishes what is old and conventional.

Normally we think of this process in terms of pronominalization, but really a pronoun is just the limiting case—a pointing mechanism with minimal semantic content that can and should be used wherever pointing is sufficient. There are many other possible realizations, especially in situations where a pronoun would be ambiguous or difficult to interpret. These include dropping modifiers, using demonstrative pronouns ("*this*" and "*that*"), and various pronominalizations of "sense", all of them markedly less informative than their original references.

The link to the pronominalization decision If you were to look back to the flowchart on page <flowchart_of_realization_procedure_n.y.w.>, you would see a decision-point in the subsequent reference module labeled "is a pronoun possible" and referring to the process described in section pronominal_subsequent_reference_n.y.w.. The "yes" path from that decision leads to pronoun selection and thense out of the realization process, while the "no" path—the one that we are interested in—leads to a box labeled "select subsequent reference strategy" and thense into the "main stream". In other words, non-pronominal subsequent reference strategies are considered only after the use of a pronoun has been ruled out, and then as a kind of "filter" on the main stream, i.e. "initial reference", realization process.

The selection of a non-pronominal subsequent reference depends on the analysis made during the pronominal part of the process. That process, to review, runs as follows: The entrance condition into the overall subsequent reference procedure is defined by set of easily evaluated predicates (pg.<decision_to_think_about_making_a_decision>) that boil down to determining (1) that the msg-elmt has passed though the realization procedure at least once before (i.e. it has a discourse history) and (2) that it is the kind of object that can be pronominalized. That is followed by a "fact-finding" period during which a set of features is compiled to describe the relationship between the current instance of the message element and the previous ones. Heuristics couched in terms of that description are then used to actually make the pronominalization decision. Should that decision be "no", we are shunted to the non-pronominal part of the process.

Subsequent reference strategies The form that a non-pronominal subsequent reference takes depends upon two things: (1) the reason why the message element couldn't be pronominalized, and (2) the way it was realized initially. The first kind of information can be read almost directly from the list of pronominalization heuristics that were applied (particularly those that voted "against"). The second kind comes from the element's history. In MUMBLE at the moment, there are so few strategies in use that the two information sources are accessed and evaluated in one operation by an ad-hoc decision tree. When the "grammar" of subsequent reference has been elaborated, a more systemic, declarative representation will presumably be possible.

Consider the excerpt below from the barber proof. Here the logical constant G, realized initially as "*Giuseppi*", appears seven times as indicated by the numbers.

> *...Call him Giuseppi$_{(1)}$. Now, anyone who doesn't shave himself would be shaved by Giuseppi$_{(2)}$. This would include Guiseppi$_{(3)}$ himself. That is, he$_{(4)}$ would shave himself$_{(5)}$ if and only if he$_{(6)}$ did not shave himself$_{(7)}$, which is a contradiction.*

Pronominalization is blocked in the second instance because another object was in focus (the "*anyone...*"), and in the third because of a likely ambiguity with that same object. In both cases, the elmt-instances for G were passed on to the non-pronominal section of the process with the reason for their "non-pronominalization" given as "ambiguity" (obviously a gross simplification). This reason was then combined, trivially it turns out, with the possibilities given by the entry for realizations of G intermediate between the initial one and a pronoun. Of course there are no intermediate forms between a name and a pronoun (at least not in English), and all that can be done is to again use the name. (This is recognized directly from the entry—its will-be property is proper-name.)

When the entry has more possibilities and the history indicates that the initial instance was a "noun phrase with modifiers", then ambiguity can be countered with a very productive strategy that is more interesting for present purposes. That strategy is to omit all of the modifiers initially used, leaving only the head noun (describing the object's class) and some definate determiner. There was an example in the section on pronominal reference (pg.167) taken from the tic-tac-toe domain where the initial reference was "*the line opposite yours*", and the subsequent reference "*that line*".

The mechanics of this strategy involve a *meta-decision*: The subsequent reference routine applies a special choice to the elmt-instance it is given. The choice edits the decision-to-make property of the instance, in this case removing all of its decisions except the matrix and the head and substituting a special determiner decision to introduce a demonstrative.

```
(define-choice that-head (instance)
    element-returned instance
    marking-actions ((edit-decisions-to-make instance
                                            retain '(matrix head)
                                            add '(definite-determiner))))
```

Presently in MUMBLE only the demonstrative "*that*" is ever used because I do not understand the distinctions involved in deciding between "*that*", "*this*", and "*the*" well enough to write a decision that any of the micro-speakers could motivate. For the first version of the logic domain, Filmore's distinction between "*this*" and "*that*" was tried, i.e. use "*this*" when the audience and the speaker share the reference, otherwise "*that*" [filmore_dexis_2]; but the domain was not really up to it.

Requirements for having subsequent reference rules

Non-pronominal subsequent reference can be understood as a set of conventions governing the ways in which the amount of information conveyed with a reference can be minimized while still making identification possible. Strategies for this include pronominalizing the descriptions from earlier instances or just leaving facts out. A prerequisite to any formalization of this minimization process is a uniform grammar for decisions, i.e. a conventional nomenclature and semantics. That is, the sophistication of the job one is able to do in a general-purpose subsequent reference procedure depends critically on the extent to which the dictionary (and the discourse history) follows some set of linguistic, pragmatic, and intentional principles. Below is an example of the kind of facility that can be developed and shared between domains as part of an interlingua.

A general technique for filtering entrys by "given" If the dictionary designer holds to certain conventions, it is possible to do the kind of "per-entry" analysis of given information discussed in section given_new_n.y.w. as a general part of the subsequent reference routine. The requirements are three: (1) the facts that each entry might mention must be organized one per decision (because the filtering will be done at that level); (2) each decision must identify the fact it is responsible for in some way, e.g. as a conventionally labeled access-expression; (3) the region of the discourse over which the test is to apply may not be idiosyncratic.

An entry that meets these criteria is role_entry below from the KLONE-nets-as-objects domain. It was responsible for all of the descriptions of roles that appear in the text on page <klone_example_n.y.w.>.

```
(define-entry role_entry (role) (matrix default (genitive-np)) (role-ness variables ((fact !(isa
,role role)))
(proper-name)
default (modifiers_gets name matrix)) (concept
default (of-gets concept)) (determiner
default (head-gets role matrix)) (name features
variables ((name nil) (fact !(name-of ,role >name)))
variables ((concept nil) (fact !(has-role >concept ,role)))
default (determiner-gets the)))
```

The general-purpose filter acts by computing the "fact" for each decision, then checking the discourse history to see if it has been already mentioned within the "non-stale" region of the text. (In that domain, this region was defined as the current paragraph, plus the last sentence of the previous paragraph if the current position was within the topic sentence.) If the fact had been mentioned, the decision could be omitted, i.e. edited out by the meta-decision, with the restriction that something must remain. As writen, the restriction was just an ad-hoc listing of the allowable minimal combinations of decisions (e.g. "matrix + role-ness + determiner" or just "name"); in the long run, this restriction will likely be found to be related to the patterns of decisions that can be used in subsequent reference strategies.

7.2 Subsequent descriptions

To the extent that a "description"—"thing described" distinction is maintained in the message expressions, it can be detected and realized by the subsequent reference routine in terms of *pronominalization of sense*: phrases like "*one*", "*another*", and "*such*". The organization of this procedure is identical to that of "normal" pronominalization, except, of course, for its realization choices. We enter the subsequent reference routine at the same place: the inexpensive tests; these use the interface function elmt-reference-type to detect that they are dealing with a description rather than a reference, causing a dispatch to a special routine for pronominalization of sense.

Among the micro-speakers this distinction was not common: the logic domain did not have it at all, nor did KLONE-nets-as-objects; with the digitalis advisor or the Macbeth domain it could be computed with some effort and search. Only in tic-tac-toe (or the real version of KLONE) was the needed information directly available as part of an object's type. As a consequence, there has been little practice in realizing subsequent descriptions—the analyses have been long in alternatives and short in distinctions. About all that appears certain is that alternatives for realizing subsequent descriptions within noun phrases are intimately related to the alternatives within clauses: the ellipsis phenomena of section <ellipsis>—they are intertwined within the same decisional systems (pg.<coordinating_alternate_ellipsis_strategies>). Consequently one needs to be able to describe the alternatives—design their choices—in such a way that they can be planned some time in advance of when the controller actually reaching the nominal msg-elmts affected; we will take this up in a moment.

Consider the simple case of:

"*You took a corner and I took one too.*"

The message-level sources for these two "corners" are shopping lists:

```
(describe 'square-1    ;i.e. the upper left-hand corner
          '((corner square-1)))
(describe 'square-3    ;upper right-hand corner
          '((corner square-3)))
```

The shopping-list format distinguishes the specification of reference (the first argument to

describe) from the specification of descriptions (the second). Each message element in the description specification (here there is only one) is entered into the discourse history just like the references. In this case the first instance of the corner relation will lead to a record describing the location where it was realized, especially that it was in the head slot, and that it was realized as a non-function word.

When the second instance of corner is reached at its own head slot later in the text, a feature-based analysis of the two positions is made, just as with references. The heuristics that access the analysis are also of the same design as for references. (In MUMBLE at the moment they are identical!). The only adjustment is to the interface function that acts as the oracle to determine whether potentially distracting descriptions noted on the basis of their syntactic position really are such pragmatically. In this example, there is no distractor, the two descriptions are in parallel positions in conjoined clauses, and the anaphoric relation is certainly not stale, thus the pronominalization goes through and the English pronoun for noun phrase head's, "*one*", is selected as the realization.[19]

The key to a sucessful subsequent description facility is a rich network of generic concepts within the expert program. Consider that a few paragraphs ago I wrote "*...the first argument to describe, ...the second*". For such a reduction to go through in this linguistic component, there must be *a single message element* representing the concept: "argument<s> to describe". Otherwise we would be required to search the history of the multi-level composite message structure that was the source of the ongoing noun phrase in order to recognize a recurrence of the sequence. Such searches were experimented with in an early version of MUMBLE, and it was found that an incredible complication of the discourse history was required to make even the simplest cases efficient. When you then see that modern-day knowledge representations all use generic networks ("AKO" hierarchies) as their basic elements—automatically supplying the needed unit objects to represent composite actions—you have a strong motivation to eliminate the search entirely and just let the less sophisticted domains do without.

A notion of "minimal NP remnant" operates with subsequent decisions just as it does with subsequent references. If there is there is some prenominal modifier remaining beyond the common generic part, especially if it is a point of contrast like "*second*" was above, then the [head] can be left empty. Of course even in such a case the [head] can still be filled with "*one*"; this is another of the subtle, poorly understood stylistic alternatives of English.

Deducing description from structure The barber proof (pg.<barber_proof>) was produced from a representation that has no reference—description distinction, the predicate calculus. Yet it

19. When the pronoun "*one*" is used and there are no pre-head modifiers, the otherwise ubiquitous English determiner must not be present, e.g. "*one who is wise in the ways of the world*". This detail is encoded into the subsequent description routine as a contingent reaction to the "pronominalize the [head]?" decision.

includes descriptive pronouns like "*such*" and "*else*". How? The answer is by (laboriously) interpreting structural facts about individual elmt-instances and including them in the discourse history—in effect manufacturing the needed distinctions solely within the linguistic component.

Consider the use of "*such*". This pronoun replaces all of the modifiers of a "noun phrase with modifiers", leaving only the determiner and the head, e.g.

> *"Assume that there is some barber who shaves everyone who doesn't shave himself (and no one else). ...Therefore, it is false: there is no such barber.*

In both instances of the message element for "the barber", the configuration of attachments on the elmt-instance for the existentially quantified variable was identical at the point when it was realized. Specifically it was:

ELMT-INSTANCE-#
real-msg-elmt X
entry-for variable-entry
entry-arguments-for X
decisions-to-make (matrix head determiner)
;n.b. these have been shifted, see pg.<variable_described_as_name_n.y.w.>
attachments (qualifier formula87)

The subsequent reference routine notices the second instance of X. The first instance is six sentences back, much too far for a pronoun but not too far for some intermediate subsequent reference strategy. The several strategies given above do not "fit" well because of the ad-hoc way in which the qualifying formula is associated with the instance (in one of the richer domains it would have been added to a shopping-list); instead, a special-purpose subroutine was written to scan the properties of the elmt-instance and collect any attachments or new decisions known *a priori* to be modifiers or qualifiers. This is then compared with a similarly collected list included in the record of the last instance. In this case they match entirely, which is a precondition for the use of "*such*".[20]

The "*else*" phrase: "*...shaves everyone who... and no one else*" is not properly formalized at all in MUMBLE. It was encoded directly as a unitary choice, an alternative way to realize the "only if" aspect of the biconditional. That may well be all that needs to be said about how to express the

---

20. The other precondition is that the noun phrase must be based on a generic description. That is determined from the idiosyncratic fact that predicate-entry was used and that it invariably deals with generics. Facts like these fall out explicitly from the message-level representation of richer domains, but must be pieced together *ad-hoc* in simpler ones that have been "augmented" by their dictionaries.

The intriguing syntactic properties of "*such*" have not been throughly analyzed here. The word is analyzed as a function word, i.e. it is taken to not ever occupy a slot in a noun phrase but rather to be associated as an attachment under the control of grammar-routines of the [determiner] which react to whether the determiner is "*no*", in which case they produce "*no such barber*"; otherwise they produce "*such a barber*". I have no doubt that there is a more integrated analysis to be found with more study.

complement of a set using a pronoun, but one would presumably like to be able to decompose the phrase into pronoun and "*else*" in a way that would permit more general rules to select it. The constituent-schema used was the same as for "*Giuseppi himself*".

7.3 Planning subsequent descriptions

Thus far I have talked about subsequent reference as though it functioned totally on a "standing order"/"targets of opportunity" basis only, i.e. every reference and description that passes through the realization process is monitored; subsequent reference strategies apply in every case possible, and take effect immediately. This is true but missleading. One does want a background process that will realize general linguistic coherency relations automatically without the speaker needing to think about it. However, the best texts, those most effective rhetorically, are the result of thorough planning[21] all the way down to the level of subsequent reference strategies.

The planning has two aspects: the specification of clausal choices (or larger) that involve specific subsequent description patterns, and the specification of the contrasts or of specific kinds of descriptions to be used across a set of msg-elmts. The first aspect is straight-forward: we want to be able to write choices that have sufficient depth as to differentiate say "*You took one corner and I another*" from "*You took a corner and I took another one*", so that they can be selected as integral combinations rather than produced only through the chance coincidence of the output of several synonym-sets.

This is simple to do on a case by case basis, the question is how to do it with some generality. Case by case you would write a new choice for each combination of (a) the syntactic structure of the two matrix clauses, (b) the clause-level ellipsis desired, (c) the location of the contrasting noun phrases within the matrixes and (d) the desired pairs of nominal descriptions. This would lead to an enormous blowup in the number of choices in the grammar and in the end would be awkward to write decisions for. A far better technique—if the speaker's messages can support it—is to factor out at least the matrices, and to establish them all as distinct decisions. This can be done by using a default-decision such as used for operators like negation or contrastive-polarity (pg.<emphasize_polarity>), or better by including the coordination goal(s) as part of a shopping list. Such a shopping list might look like this:[22]

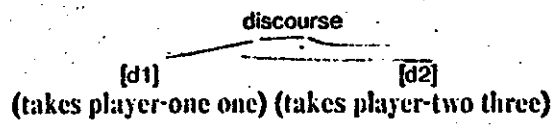
---

21. I do not mean to suggest that the space of alternatives is explicitly reasoned through on each occasion. More likely, it seems to me, is that "planning" is done in terms of the selection of large, highly parameterized plan schemata that are selected on the basis of a small alternatives space involving relatively little deliberation. phrases are perfect example of plan schema at a linguistic level as they (1) define procedures, and (2) are parameterized by msg-elmts and by grammatical context. I suspect that full-scale "planning" only takes place incrementally over time through adjustments to the schema or to the criteria in the alternatives space, and should be thought of as a kind of learning.

22. This is taken from the fledgling tic-tac-toe domain. The proper level of conceptual abstraction to use here—i.e. what the tic-tac-toe kibitzer actually notices such that leads to a text like this—is still being experimented with.

```
(discourse-unit '((sequence  (takes player-one square-1)
                             (takes player-two square-3))
                  (describe 'square-1
                             '((corner square-1)))
                  (describe 'square-3
                             '((corner square-3)))
                  (coordinate  sequence '(emphasize corner)) ))
```

In this shopping list, all of the planning to pick out a subsequent reference strategy has been gathered into the last item ("(coordinate...)"), which dictates that the sequence of the list is to have some coordination strategy applied to it such that concept corner is emphasized. The entry for coordinate applies after the other three items, at which point we have:




```
elmt-instance
      real-msg-elmt (corner one)
      entry-for  corner-entry

elmt-instance
      real-msg-elmt (corner three)
      entry-for  corner-entry
```

Against this background, the entry for "coordinate" selects the "one-class-and-gap-another-blank" choice (below). At the moment, it is selected just by an association with the key-word "emphasize" and the knowledge that corner will be realized as a head noun (which it gleans from corner's entry). Behind this association (but presently known only to the designer) is the notion that a term can be emphasized by leaving it out in a place where it is predictable. Note that this choice functions entirely by side-effects

```
(define-choice  one-class-and-gap-another-blank  (inst-1 inst-2)
    actions  ((preempt-decision [mvb enter-slot]  ;whose decision to preempt
                                'do-gapping-if-possible   ;name of decision preempted
                                '(gap)  ;CHOICE it is to now use
                                '(member 'd2 'vertical-context))  ;text to specify which instance of the decision
              (preempt-decision  inst-1
                                 'determiner
                                 '(determiner-gets  one))
              (add-decision  inst-2
                             'modifier
                             '(modifier-gets  other))
              (preempt-decision  inst-2
                                 'head
                                 '(do-nothing)) ))
```

Comparable choices would to pick out others of the many other coordinating combinations that are possible.

The only point to having choices like this one is that there are specific reasons, known early at the level of the containing conjunction or discourse node to the effect that one or another combination of effects is preferable to those would be selected automatically by separate later decisions made much closer to their points of effect. If the lowlevel decisions are just random selections from a synonym-set (as they are at the moment in MUMBLE), then they will probably not be superior to a highlevel decision, even for all the trouble it can be to write them.

7.4 Coordinated references

The last example involved a coordinated reference: the realizations of the two instances of "corner" were fixed in advance of either being reached by the controller, and as the result of a common choice. Coordinated references are important to the design of the linguistic component because of the restrictions placed on them by the stipulations of the theory, i.e. all texts are produced in one indelible realizing pass top-down and left-to-right through the tree. This means:

(1) Unless marked for coordination by an earlier decision that dominates both instances, the first instance reached will have no idea that there is a second to follow and will select its realization freely without any thought to coordination.

(2) Once the second instance is reached, the need to coordinate or distinguish the two could be noticed. However, since the process is indelible—the controller is not about to backup—there is no way for the realization of the first instance to be changed retroactively.

Certain kinds of distinctions can be noticed and implemented "on the fly". One could, for example, have a standing order that objects with the same generic description were to be distinguished whenever found within a certain distance of each other in a text, e.g.

"*First I picked up a green block and put it into the box, then I picked up another one.*"

This can be done as part of the subsequent description routine. Here, after noticing a second occurance of the description "green block", it adds a modifier decision to include the word "*other*".[23]

If we wanted instead to leave a standing order that, say, successive blocks were to be constrasted according to the set membership, e.g.

"*...one of the green blocks ...and... the other one*"

we would be unable to implement the order unless we were prepared to have every issolated block come out as "*one of the ...blocks...*", which I take to be excessive.

Instructions for contrast Every coordinated description must be indicated as such either in the

23. The Blocks World, like the tic-tac-toe domain, uses a default-decision to choose its determiners (see the system on pg.<determiner_system>) which, in this case, will select the determiner "*a*" because there is more than one object in the Blocks World that matches the description "green block". The morphology routine will then adjoin the "*a*" and the "*other*" to form "*another*".

message or as a decision made some time before the description is to appear in the text—there is no alternative in this theory if the texts are to have any variety. One might argue that this restriction causes a problem—that it is evidence that the indelibility stipulation should somehow be lifted. I would reply that it merely indicates that the linguistic component is not able to perform miracles, and I would further argue that the information needed to notice a contrast is easier to collect and reason about at the level of the speaker and in the expert's representation than online within the linguistic component (i.e in the midst of realizing a message) and using extended surface structure.

In a domain where the speaker arrives at messages by building them up from smaller propositions and relations (e.g. tic-tac-toe or the real KLONE) the speaker will inevitably "touch" all of the elements involved in any coordinated description. This puts it in a natural position to record, then and there, any rhetorical goals or relation that tie those elements together and might motivate their coordinated description.

Consider the description of moves in tic-tac-toe. A move can be described at many many levels, from the position of the square taken to the move's function in a strategic plan. Reasoning about these possibilities and their ramifications for the style of a text is the largest part of the tic-tac-toe kibitzer's job, and is presently formalized in terms of a hierarchy of entrys that dictate the structure of the next level down by selecting among alternate shopping lists. For example, if the tic-tac-toe kibitzer wanted to realize some observation only at the level of threats and counters, it would specify only that level in the shopping list, including those identifying properties that are appropriate to that level and leaving out the others, as below. The instructions for contrast are thus given explicitly by the speaker in terms of what it has included and excluded.

```
(describe 'event-2
          '((countered move-4 move-3)
            (describe 'move-4
                      '((threat move-4)
                        (player move-4 player-one)))
            (describe 'move-3
                      '((threat move-3)
                        (player move-3 player-two))) ))
```

This message element will be realized as:

"*Your threat countered {mine / my threat}.*"

The only remaining free stylistic parameter ("*mine*" vs. "*my threat*") is, not coincidentally, one that can be decided *in one pass* by a default-decision within the subsequent description routine.

7.5 Predictable facts

So far, all of the subsequent references have been decidable just by using criteria that the lingustic component could easily keep track of, i.e. the identity of the msg-elmts that have been realized so far. However, there are other dependencies at work in natural language besides just identity with a previous instance: there is an information flow from the speaker to the audience and set of inferences that the audience can be expected to make based on the pragmatic content[24] of what they hear. My first experience of such inferences with MUMBLE involved a text from a "chess" speaker with which I was dabbleing before leaving it for the easier domain of tic-tac-toe. How do we write an entry for chess pieces that can say:

"*The black queen can now take a pawn.*"

The situation is analogous to the others: a property in the first reference to a chess piece, namely its color, has been left out in the second reference because it is redundant. The problem is that the linguistic component, being what it is, has no idea of what pragmatic inferences an an audience will or will not make, and thus is blind to the redundancy. We could, of course, incorporte a specific check into the entry, but the question is what can it look for?

The trigger above was the use of a "color-polarizing" verb: "*take*"; but there are too many such verbs to imagine listing them in the entry, and there are too many syntactic nitches where the verb (or adjective, or...) might be located to imagine keeping track of them just for the benefit of one entry. To cut down this diversity, the trigger should be posed at the message level: we define a concept of "color-polarizing" relation within the expert program, and, provided the message representation is suitably rich, define it in such a way that the fact that some relation is a color-polarizer will be part of the message whenever the relation is and can have its own entry or default-decision.

The next ingredient needed is some cannonical "place" for the entry to look to see if color-polarization is in effect. The most appropriate representational device to use appears to be an ad-hoc region-feature (i.e. one that would be special to the dictionary used rather than part of the general purpose grammar). It would be set by the entry for "color-polarization", which would associate it with the region of the tree defined by whatever was the current node of the tree at the time. It would, of course, then become undefined when that node was left.

The final ingredient is some means of telling which instances of chess pieces are "subsequent" with respect to this color-polarizing relation. If the scope of the relation (i.e. the domain of the region-feature) corresponds to one of the already defined discourse regions such as the current clause, current sentence, this-and-the-last sentence, or the current paragraph, then

---

24. The rhetorical, stylistic and intentional content (in so far as the audience can guess it) are surely also the basis of inferences, probably involving emotional and interpersonal models. They are, however, too subtle to attempt to formalize now.

"not being the first piece" can be defined as being in a region (of the appropriate kind) where the concept of chess piece has been mentioned.

This kind of analysis of the mechanism by which pragmatic inferences influence production leave the door open for a vast number of pragmatically motivated region-features to flood the tree. This may not be wrong, but it is surely unaesthetic. What it suggests as an alternative that "discourse scope" is defined over more than just linguistic objects. It suggests that the computational context on the speaker's side of the line—the context where the predicates of entrys are evaluated—is a tree issomorphic to the one on the lingustic side, though probably with less detail. This is a question to study in the future.

CHAPTER SEVEN

Appendices

1. The Program

1.1 History

The basic design of the linguistic component was developed during the 1974-1975 academic year and was described in my Master's thesis [ddm_masters]. The design described there has not changed in its basic points: a grammar as a library of choices with both declarative and procedural aspects, the direct control of the process by a non-linguistic message, the use of a dictionary to decompose messages into their shared elements, dictionary entries as context-sensitive decision-procedures, and the construction of a surface-level constituent structure to control the process and provide a linguistic context for the entries were all present in some form.

The implementation was begun during the summer of 1975, and the first texts were produced the following spring using short, ad-hoc messages inspired by the Personal Assistant project. The first micro-speaker, the logic domain, was developed during 1977, and occasioned a complete revision of the linguistic side of the implementation; the "barber proof" used as the development set for that effort was completed in December 1977. Several conference papers were written at that point describing the interleaving of the realization decisions and the grammatical processing [bergen], the treatment of reference [tinlap], and the use of an interpreter to introduce conventional decisions and grammatical filtering into the dictionary [toronto].

During the 1978-1979 academic year, the design of the dictionary was totally revised, leading to the schematic format with multiple interpreters that is presented here. Also during that time, the work on other micro-speakers was begun: Ken Church developed a preliminary dictionary for Swartout's Digitalis Therapy Advisor during the fall of 1978; the KLONE-nets-as-objects dictionary was written the following spring, and the Macbeth domain was written in July 1979. The program has not been worked on since July 1979. All the examples reported on here were either running at that time or were simulated by hand on the basis of the existing grammar and dictionaries and straight-forward extensions.

With the conclusion of this thesis, a third major revision of the program is planned. The goals of that project will be:

- ☐ projection back into the implementation of the changes in terminology and analyses that were made in the course of writing this document;
- ☐ standarization of the implementation of the various data-types with the intention of making it possible to move the program to different species of LISP (the present implementation reflects four years of changing conventions);
- ☐ incorporation of on-line cross-indexing information for the grammar and dictionary with dynamic error-checking to facilitate the extention of the program by people other than its author.

I.2 Program Statistics

Programming language MUMBLE is written entirely in MACLISP [moonual], the version of LISP developed at MIT for the PDP-10.

Size The file space required to store the ASCII text of the program with its in-line commentary is distributed as follows: (in thousands of PDP-10 words)

Basic system:		
Defining all data-types		24.8
Controller & Entry-interpreters		8.7
Extensions to LISP		5.5
Debugging aids		9.8
Grammar	32.3	
Dictionaries		
Logic	11.2	
KLONE	7.6	
Macbeth	3.9	

When loaded into an otherwise empty LISP (size 57k), MUMBLE proper occupies an additional 40k words (uncompiled). The dictionary for the Macbeth domain, after expansion by the entry-postprocessor, adds a further 30k words.

Execution speed At this writing, only interpreted versions of MUMBLE have ever been tested. (The frequency with which the code was augmented and edited made incremental compilation using the PDP-10 too awkward to be profitable.) Also the entry-compiler has not been implemented. With these handicaps, MUMBLE has produced text at an average of about three seconds per word. (The median speed is more like a half-second per word given the large delays incurred at major discourse boundaries while the text is planned.) The conventional wisdom is that compilation alone will cause a a factor of ten speedup.

2. Their representations and interfaces

Beyond the fact that they are all implemented in LISP, the message representations of the micro-speakers have very little in common. Following the categories developed in chapter four, we can make the following summary:

- ☐ The logic domain: self-defined objects in issomorphic messages.
- ☐ KLONE-nets-as-objects: a mixture of self-defined and contextually-defined assertions in shopping-list messages.
- ☐ The Macbeth domain: contextually defined objects in shopping-list messages.
- ☐ The Digitalis Advisor: twenty-questions objects in issomorphic messages.

In this appendix, I will go through each of the domains in turn, outline their representations, and show how that determined the form of their interface functions.

2.1 The logic domain

Unlike any of the other domains, the implementation of the logic domain was designed with only the convenience of the linguistic component in mind. Let us consider what this convenience amounts to. Messages in the logic domain are well-formed formulas (or proofs) in the predicate calculus. They are issomorphic to their realizations, i.e. every subexpression in the message formula will have a corresponding English phrase and no phrases will be used that do not correspond to some part of the message expression as passed to the linguistic component. This means that we should use a representation for formulas that is easy for the linguistic component to manipulate and that gives it ready access to the particular information it needs.

What the representation must include

The only "manipulation" that the linguistic component does with a message element is (1) to ask various questions of it, and (2) to construct elmt-instances for it and to position these as constituents in the tree. The minimum requirement then is that there is some way to "name" the element so that it can be given as an argument to an interface function or be remembered as a property in an elmt-instance, and that this name will continue to access the same object within the domain for the entire life of a discourse. This is accomplished in the logic domain by creating a permanent, distinct LISP object to represent each wff and its subexpressions (see below), and using a LISP pointer to the object as its "name".

What information does the linguistic component need? Basically it needs to know only one kind of thing: "what entry is it expected to use to realize a particular wff?". Thus we make that fact one part of any LISP object that denotes a wff. Next, as this is a domain of issomorphic messages and practically every choice will involve embedding logical subformulas into a just-

selected, linguistic matrix, the most important thing that the entry needs to know is what a wff's subelements will be. We make that answer easy to determine by making those subelements also an immediate part of each wff's representation.

Another category of information that some entrys will need is a means of answering interface functions such as clmt-pluralp or clmt-gender that depend upon "extra-logical" information. Since in this logic domain there is no expert program or knowledge base that might also use such information or from which it might be derived, the simplest way to supply the answers is provide a feature-list as part of a wff's representation, and to define those interface functions directly in terms of features like male, or plural.

The final category of information is "identity" information: are two expressions, appearing, say, in different lines of a proof, two instances of the same wff? This a complicated question. We presumably want two wffs that are lexically identical except for the names of their bound variables to be considered completely equivalent; yet do we want the two species of variables to pronominalize each other? The tact I have selected is to answer yes and no respectively. Every instance of a quantifier defines a "new" variable that may have multiple instances in the rest of that wff. The tricky part comes when a variable is used as the basis of a generic, e.g. "*some barber*". Thus far, the tack has held up; it seems possible however that some proofs will call for subsequent references to these generics across formulae, in which case a more sophisticted definition of "previous instance" will be required.

The representation

People typically deal with wffs as character strings. Doing the same here, however, would mean including a parser with each entry in order to pick out subexpressions. Instead, a formula that the user wants MUMBLE to process is parsed only once, as it is typed in by the user, and the results are captured in a set of records[1] that is organized into a tree structure exactly mirroring the logical structure denoted by the original character string. Below is the set of records that represent the first line of the barber proof.

1. To be precise, MACLISP "hunks"—contiguous blocks of memory that are accessed by a numerical index. Property-lists or ordinary list-structure would have done just as well.

Line 1: $\exists x$ (barber(x) $\wedge \forall y$(shaves(x,y) $\leftrightarrow \neg$shaves(y,y))) premis

```
line1     = [ line . () . formula89 . premis ]
formula89 = [ formula . () . conj88 . (x1) ]
conj88    = [ conjunction . () . 2 . (pred82 formula87) ]
pred82    = [ predication . () . barber . (x1) ]
barber    = [ name . () ]
x1        = [ variable . () . formula89 . x ]
x         = [ name . (male person) ]
formula87 = [ formula . () . iff86 . (y1) ]
iff86     = [ iff . () . pred83 . neg85 ]
pred83    = [ predication . () . shaves . (x1 y1) ]
shaves    = [ name . () ]
y1        = [ variable . () . formula87 . y ]
y         = [ name . (male person) ]
neg85     = [ negation . () . pred84 ]
pred84    = [ predication . () . shaves . (y1 y1) ]
```

For ease of manipulation, each record is made the value of a LISP atom (**line1**, **formula89**, etc.), whose name is then used as the name of the record. The equivalence of two lexical expressions is determined at the time they are typed in, and is reflected in the reuse of the same record as shown above for the variables. The first field of a record is always its object's logical *type*, and the second is always its feature-list; the rest of the fields vary from type to type.

Linking wffs to their entrys

The dictionary for the logic domain is based on common entrys for the logical vocabulary and individual entrys for particular predicates, variables, and constants. Thus the first step in linking wffs to their entrys in this domain is always to look at the type of their record. If it is part of the common vocabulary, then the type itself tells us which entry to use. If the type is name, the catchall for individuals, then we know to find the entry directly associated with the individual.

Exactly how the "association" is performed is a matter of preferred programming style. The function below was the one that was actually used: it uses table lookup for the logical vocabulary (largely because the number of items is small and fixed), and a direct association by attached properties for the individual. When the dictionary itself defines the properties (i.e. for "marked-to-play-a-role"), an additional indirect step is required. Entry-for returns the name of the entry it finds.

```
(defun entry-for  (exp)
    (cond ((and (boundp exp) ;first test that it is the correct kind of LISP object
                (atom exp)
                (hunkp (symeval exp)))
           (cond ((get exp 'marked-to-play-a-role)
                  (let ((role (get exp 'marked-to-play-a-role)))
                      (get role 'dictionary-entry)))
                 ((eq (type exp) 'name)
                  (get exp 'lexical-entry))
                 (t (cadr (assoc (type exp)
                                 ((proof  proof-entry)
                                  (paragraph paragraph-entry)
                                  (line  line-entry)
                                  (formula  formula-entry)
                                  (neg  compose-negation)
                                  (predicate  predicate-entry)
                                  (if-then  if-then-entry)
                                  (iff  iff-entry)
                                  (conjunct conjunct-entry)
                                  (disjunct disjunct-entry)
                                  (variable variable-entry)
                                  (proposition  proposition-entry)
                                  (constant  constant-entry)) )))))
          (t (print '"entry-for: don't know how to find an entry for EXP")
             nil) ))
```

The clause of the conditional that tested if exp was "marked-to-play-a-role" is part of the domain's facility for labels, and will be discussed later with the labeling facilities of other domains.

Other interface functions

There are no surprises with these functions: all of the potential difficulties of definition have been relegated to the wff parser, or are ignored entirely and supplied *ad hoc* by the designer.

entry-arguments-for Trivially defined to be the real-msg-elmt of the elmt-instance being realized.

msg-elmtp Equivalent to entry-for, which returns nil if the object has no entry.

same-msg-elmt Since all message elements are LISP atoms, this is just the LISP eq.

elmt-pluralp/elmt-gender Usually, this information is just read out from the features field on the message element, but see below.

elmt-reference-type Not used.

elmt-discourse-history Since the "speaker" is trivially in the same computational environment as the lingustic component, the history is maintained as (1) a chronologically ordered list of logical objects realized, plus (2) "discourse-history" properties on the individual objects summarizing their own chronologies.

distinguishable-kind Not used.

Artificially augmenting a domain's concepts

Any attempt to give the logic domain a greater fluency with English quantifiers is immediately confronted with the fact that the act of making the decision between "*all*" and "*each*" (or "*any*") adds information to the logical formulas that was not there to begin with, namely whether the exemplar being used for a given quantifier is plural or singular. Once decided, this information must be remembered, and remembered in way that does not change the original formula.

The technique used was an obvious if inelegant one: an association-list was maintained where formulae whose quantifiers had undergone one of these decisions were stored, paired with a record of what had been decided. Then the interface function elmt-number was written to look first to this list of specializing information before going to the logical formula itself.

The fact that the predicate calculus supplies only two quantifier symbols while English supplies (at least) four, is not a deficiency of the underlying representation so much as it is an indication that a special body of rhetorical criteria will be needed to decide how to pin down the extra descriptive dimensions available in natural language—richer intentional motives are needed, not necessarily a richer semantics. In general, deciding between a singular or a plural version of the universal quantifier is non-trivial and often can't be done in one pass. In this sentence (taken from earlier in the paper), I had originally used the plural form (given first in the pairs), but went back and changed what I had written to use the singular form because that should (I reasoned) reduce the potential for a distributive/collective ambiguity in the final phrase "*their total votes*".

> "*(All/each) of the distractors (are/is) run through the pronominalization heuristics as if (they/it) (were/was) the current-instance and their total votes compared*"

2.2 KL-ONE-nets-as-objects

The dictionary and interface functions of this domain are not designed to process actual KL-ONE-nets. Instead, the network to be processed is first passed through an interface program that *transcribes it as a set of assertions.* (The diagram below shows the assertions that were created to describe the pp concept shown in the figure on page <example_kl-one_net>.) These assertions follow the usual syntactic conventions of pattern-matching languages such as MICROPLANNER [microplanner_manual]: list-structures with the name of the relation first followed by its arguments, and were indeed accessed by their entrys via pattern-matching.

```
(subconcept :c:pp :c:phrase)
(has-role :c:pp :r:pobj{pp})
(has-role :c:pp :r:prep{pp})
(has-role :c:pp :r:interp{pp})
(has-role :c:pp :r:ppobj{pp})
(value-restriction :r:pobj{pp} :c:np)
(value-restriction :r:prep{pp} :c:prep)
(value-restriction :r:interp{pp} :c:relation)
(value-restriction :r:ppobj{pp} :c:pp)
(subconcept :c:ofpersonpp :c:pp)
(subconcept :c:insubjectpp :c:pp)
(subconcept :c:locationpp :c:pp)
(subconcept :c:aboutsubjectpp :c:pp)
```

The initial motivation for this "intermediate" assertion language was technical convenience. Because of the limited address space of the PDP-10, the KL-ONE programs and MUMBLE were forced to be resident in different forks and able to communicate only by external files—a translation of some sort was thus inevitable.

Artificial first class objects

Even on a machine with a larger address however, a shadow, assertion-based representation would have its place in this domain. The present implementation of KL-ONE has a unique internal object to represent each formal object that is at the end of a link; it does not however have internal objects corresponding to the links themselves. For example, there is no object to denote the relation (has-role :c:pp :r:pobj{pp}); instead, that information is given in a table that is part of the object for :c:pp and again in a table with :r:pobj{pp}.

This is a problem because an entry in a table is not a very versitile object: there is no way to make it part of an elmt-instance, no way to add properties to it to record its discourse-history, no way to apply an interface function to it, and no way to associate it with an entry. In short, it is not a first class object (pg.<first_class_objects>).

Since KL-ONE does not have first class objects for its basic relations it is necessary for its speaker to construct them. The assertions are stand in's for the unmanipulable primitive relations of KL-ONE nets.[2]

---

2. This assessment of KL-ONE is not entirely fair. The KL-ONE representation has explicit provisions for first class objects to represent these relations, namely individual concepts on a "meta-level". Exactly such meta-level concepts will be used in "real" dictionary for KL-ONE where the objects to be realized are not the nets themselves but the "real world objects" in the program's model that the nets represent. However, in the KL-ONE-nets-as-objects domain, one of these meta-level concepts would be needed for every relation in the net, which would be an intolerable overhead.

Associating assertions with entrys

The convention of making the name of the relation the first item in an assertion can be taken over directly into entry-for: we associate entrys with relation names, and then determine that the entry for an assertion will be the entry for its relation.

The arguments to a KL-ONE-nets-as-objects relation will always be objects from the KL-ONE net.[3] These of course do have first-class objects to represent them (they are the source of the colon-filled names print-names), but the Rubicon has been crossed, and they are represented in the assertion language by new atoms with corresponding print-names. The entrys for these atoms are associated with them at the same time they are created: the entry's name is put on their property-list, e.g.

```
|c|clause — name    [ word . (proper-name classifier) . clause ]
           type    concept
           entry   concept-entry
```

The entry-for function thus has two cases: when the object is an assertion (i.e. a LISP list) it looks up the entry on the property-list of its first element; and when the object is a KL-ONE entity (i.e. a LISP atom) it looks up the entry on the object's own property-list.

The other interface functions

entry-arguments-for Because this is an assertion-based domain, the arguments to a relation are trivially available (they are the rest of the elements of the assertion). It is thus more efficient to package them at the point when the elmt-instance is being assembled than to have the entry make a redundant access operation later.

msg-elmtp This can always be defined as entry-for(assertion).

same-msg-elmt LISP equal, which compares lists recursively at all of their levels.

elmt-pluralp, elmt-gender, elmt-reference-type Since all objects in the domain were either relations or proper names, these functions were defined to just return constant values.

distinguishable-kind Defined to be identical to the type of the object, i.e. "concept" or "role".

Representing an unbounded search in the tree

The simplest way to integrate the depth-first traversal of the concepts in the net with the process that reads them out in English would be to have the search take place in directly the the tree's computation environment and to connect the output stream of the search directly to the the tree's "next available slot for a message" (see section initialization_n.y.w.). The tree would be extended incrementally, even to the point of growing and embedding new discourse nodes automatically by

3. With the exception that when a series of assertions have been merged (see message_level_reduction_n.y.w.), the argument merged on will be replaced with a conjunction node.

enriching the vocabulary of the search processs's output to note when a new set of subconcepts was entered.

This, of course, is not possible in this case because the two processes actually lived in computationally independent forks. Instead, it was simulated by using pattern-matching within the major entrys to accumulate the assertions on demand concept by concept (see for example the concept-defining_entry pg.<concept_defining_entry>). The text was planned and realized one paragraph at a time, with the known pending concepts ("paragraphs") stored in special-purpose "state-assertion" message elements at the right fringe of the tree.

2.3 The Macbeth domain

The internal representation of the Macbeth domain is a version of the language FRL (see footnote pg.<frl_references_n.y.w.>). The primary objects of this language are "frames": multi-level, "augmented property-lists" that support default values, inheritance, and the attachment of access-triggered procedures. Within FRL, the user always manipulates a frame by using special-purpose access functions, and thus need know nothing about how a frame is constructed. The interface to MUMBLE, on the other hand, must know about the internal structure of a frame because it expects to build messages directly from frame properties and values.

As FRL is implemented, a frame is accessed by its name, e.g. macbeth or murder-ma, which is implemented as a LISP atom. The frame proper is found on the atom's property list under the tag frame; it is a list structure, and uses level of embedding to implement the frame>properties>values>facets distinction. The frame for duncan is shown below as displayed by a pretty printer.[4]

4. Note, this figure is exactly what the interface and dictionary were written to manipulate. The frames shown in the introduction were "cleaned up" to avoid irrelevant details.

```
[duncan]
    frame (duncan (part-of (value ma (see (relation8)))))
                  (ako (value (king (see (relation24)))))
                  (hq (value (dead (see relation47)))))

                  ;the frames "relation8" and "relation24" are just backpointers
[relation47]
    frame (relation47 (frame (value (duncan)))
                      (slot (value (hq)))
                      (value (value (dead)))
                      (caused-by (value (relation46))))
[relation46]
    frame (relation46 (frame (value (macbeth)))
                      (slot (value (kill)))
                      (value (value (duncan)))
                      (cause (value (relation47)))
                      (caused-by (value (murder-ma))))
```

Contextual Definition

All FRL functions deal with frames as wholes: one cannot break down a frame into, say, individual properties and still expect FRL to be able to understand them as such. Outside of their frame, individual properties and their values are just so many lists and atoms, with no markings to indicate that they have some special interpretation.

Because FRL objects, i.e. properties, slot-value pairs acting as abstract predicates, constant symbols acting as values, and frames acting as values, cannot be identified as such in issolation, I will say they are *contextually defined.* Contextual definition is an inconvenience for the interface but not an insurmountable one: whenever we want to extract one of these objects from its frame, say because we are decomposing the frame and positioning its parts in the tree, we must be sure to mark each piece (or the piece's context in the tree) in some way that will record the object's FRL-identity. Let us look at some examples of this.

When the message is an entire frame (as in the play summary, pg.<ma_frame_text_n.y.w.>), the frame is broken up by whole-frame_entry into its individual properties, and these are then positioned in the tree to be sentences in a paragraph. As extracted, these properties are the same lists that are used for abstract predicates, e.g. (ako (value (king))) or (hq (value (dead))). By taking the properties away from their context, we have removed the usual means of determining (1) what frame they were properties of, and (2) that they were frame properties and not abstract predicates or just random lists. Thus before positioning the properties, the entry adds to each list the name of their frame and the specially recognized indicator property, yielding: (property (duncan (ako (value (king))))), (property (duncan (hq (value (dead))))).

The difference between, e.g., duncan as a value and duncan as an independent frame (the first becomes a proper name and the second an entire paragraph) is handled by context, i.e. the paragraph interpretation is used only when the linguistic context—the type of slot that the elmt-instance is embedded in—is one that permits paragraph-sized structures, otherwise a summary is used. Whether the summary is to be a proper noun ("*Duncan*") or a simple clause (e.g. "*Macbeth murdered Duncan*") is determined by examining the frames (which we can do because our message element here is the atom the frame is attached to) and using a clause whenever the frame includes backpointer properties, otherwise using a proper noun formed from the atom's print-name.

All FRL objects are either atoms are lists. The atoms always stand for frames, and have just been discussed. Lists are either properties or abstract predicates; but since a property will always begin with the reserved atom property, we can safely interpret any other lists that are encountered as abstract predicates by default, and need not mark them specially.

Interface functions

Entry-for Because of its intentional resemblance to English, the internal representation of the Macbeth domain is one of the simplest to write a dictionary for. As a first approximation, there need to be only three general purpose entrys, one for each type of FRL object. These entrys extract elements from uniform positions in the objects and apply them to equally uniform choices. (See sections from_frl_to_english_n.y.w., and <default_property_entry>.)

Deviations from these defaults are arranged by associating a special-purpose entry[5] with some predictable subelement of the object, e.g. the atom denoting a frame, or the atom that names the property in a property or an abstract predicate. The "association" is implemented by putting the name of the entry on the property list of the affected atom under a tag that indicates when it is to be used (e.g. predicate-entry or summary-entry).

The Macbeth domain version of entry-for first determines which of the three kinds of FRL object it is working with, and then looks further to see if there is a special-purpose entry that should be used instead of the default.

entry-arguments-for As in the logic domain, it is the object itself. The decomposition is handled within the entrys.

msg-elmtp Again, just using entry-for is sufficient.

same-msg-elmt The LISP equal predicate.

elmt-pluralp, elmt-gender Presently objects with marked properties (i.e. plural or feminine) are given a feature by hand in the dictionary to indicate that. With some searching through the FRL network and some linguistic annotation of generic concepts, such information should be computable directly fromt the domain.

---

5. More likely it would be an entry-schema that is used, see pg.<entry_schema_macbeth_domain>.

clmt-reference-type Trivialized at the moment since there were no pronominalizable descriptions used.

distinguishable-kind Again marked by hand in the dictionary; "actor", "event", and "action" were distinguished..

3. Grammar-variables — binding discipline

CONTROLLER-VARIABLES are recursive. This means that if, for example, we want the variables *current-subject* and *current-mvb* ("main verb") to be defined in every clause, then we must have some provision to "save" the former values of these variables whenever we move temporarily into an embedded clause, for example when we enter a relative clause off of the subject NP of the "original" clause. There are several established ways to save the early values of recursive variables; I have elected to use "shallow binding".

Shallow-binding is a standard variable maintenance technique where the value of a variable is always accessed in the same way regardless of how many recursive embeddings it has undergone. Former values, values that will have to be reinstated when the program leaves the embedded region, are stored on a push-down stack. The details of the stack mechanism are not relevant here except that in MUMBLE they make essential use of the record properties on the NODEs, which are also used by the HISTORY-TAKERs as summaries for use after the actual constituent structure has been expunged.

Grammar-variables are managed exclusively by GRAMMAR-ROUTINES. Region entering routines set the variables; region leaving routines reset them to their former values. The [clause enter-node] routine in MUMBLE, for example, sets the variables *current-clause* and *current-subject.* Its counter-part, [clause leave-node], resets them to their former values. These routines avail themselves of a common variable setting/resetting protocal which gives each variable its own shallow-binding stack.

Center-embedding

Note that no way has been provided (thus far) for a GRAMMAR-ROUTINE to know how deeply or (even whether) it is embedded. Every time [clause enter-node] is activated, it can sample other grammar-variables such as *current-slot* or *current-mother-node*, but these are also deitic variables that "move" with the controller. There are no variables in the linguistic component that count absolute depth or any other absolute quantity. This "non-feature" is based on a hypothesis that such information is not needed in the grammar. (N.b. one *can* design a routine in a HISTORY-TAKER that counts. This has been done for several of the experimental

speakers as the basis of stylistic-heuristics designed to limit production with depth.)

But, while the grammar should not need to count, there may be other, more qualitiative aspects of embedding that would be relevant to it. I am indebted to Ken Church for pointing out a simple variation on the stack maintenance discipline that makes it possible to distinguish cases of left and right embedding from center embedding. This is important because while the first two cases can be parsed with a finite state device, center embedded texts require a pushdown automata¿intrinsically more powerful. Where there is an option in choice of embedding, choosing left or right over center embedding should simplify the job of the audience reading the text. (An example of a right embedded text is: *the dog that chased the cat that killed the rat that ate the cheese...*; of left embedded text is: *the dog's cat's rat's cheese...*; and of center embedded text: *the rat the cat the dog chased killed ate the cheese.*)

The variation is based on the observation that in both left and right embeddings there is no need to save the former values of controller variables because the context in which they are valid will not be visited again¿only in the case of true center-embedded phrases is a stack necessary. If there is no "former value" of the variable (which would be the case, for example, when starting an entire message), then there is no need to begin a stack. If there are no constituents remaining at the level of the embedded constituent to be returned to when it is finished, then there is no reassign the old values to the variables. Only if there are constituents of the current constituent current sentence remaining to the right of the current position and thus no yet realized is a stack necessary. The test for whether one is currently center embedded is simply whether or not the embedding variable (e.g. current-clause) has a stack.

4. The Discourse History

The linguistics component keeps a record of every realization and every grammar-decision" that is made in the course of its processing. The purpose of this record is to facilitate subsequent decisions about pronominalizations, parallel contexts, stylistic variation among synonyms, and usage options in general. For this reason, it is (1) organized as a random access store, and (2) given in terms of generic descriptions of the events: a "noun phrase" was created, rather than NP3; the choice *persuade-type-clause* was selected, rather than the choice-application (persuade-type-clause '*persuade* 'lady-macbeth 'macbeth '(macbeth murder duncan)).

One may ask why a separate record is needed when all of the information is already present in the treed (or would be if derivational annotations were added). It is because one cannot divorce a data representation from the kinds of operations that are used to access it. The information is indeed "already" in the tree, but the form that it takes there is awkward to

manipulate. Consider, there are only two ways to access information that is stored in a tree: to scan it, item by item following the constituent structure until the wanted information is found, or, having scanned it once, to explicitly remember all information that might prove relevant. The first method is not used at all in this theory because of its intrinsic inefficiency. (Recognizing the information can require parsing the tree and performing deductions, and possibly polynomial search times could be required, jeopordizing the linear time property of the overall process.)

The second method is used to good effect by the GRAMMAR-VARIABLES; however, it has limitations of distance (one does not generally need deitic information about "the fifth sentence back"), and of specificity (GRAMMAR-VARIABLES use a grammatical vocabulary: "the current *subject*". The pronominalization routine must be more more specific: "the last occurence of macbeth"). In a discourse history, where what is wanted is information about specific objects involved in past events with an arbitrary relationship to the current position, what is needed is an associative record, separate from the tree and compiled at the time the events occured.

The discourse history is compiled as follows. Whenever an event needs to be recorded, a HISTORY-TAKER operation will be applied. The HISTORY-TAKER compiles a RECORD describing the event, which it then has added to the HISTORY of the object involved. (When the object is an ELMT-INSTANCE, extra-linguistic operations may be involved to make the association between the message element, this instance of it, and its HISTORY. These are performed by the interface functions elmt-discourse-history and set-elmt-discourse-history which respectively retrieve and augment message element HISTORYs.)

Data type: RECORD

Exactly what properties a RECORD should have is a question of analysis. It is clear however that the properties will vary depending on the type of event whose history is being described. As a consequence, every RECORD will have an event-type property that will be used to determine what other properties can be expected. MUMBLE has four event-types: *msg-elmt_realized* (when was macbeth last mentioned?, how was it realized then?, what was its discourse role?); *decision_made* (did we do VP-deletion the last time we had a conjoined verb phrase?); *choice_selected* (when was the last time we selected *headK-classname* ?) and *node-constructed* (what was the focus of the last paragraph?). Because of space limitations on the program, MUMBLE only makes RECORDs of selected DECISIONs, CHOICEs and CATEGORYs for which it has specific usage heuristics.

Generally speaking, a RECORD will note the position of the event in the tree, a description of the outcome of the event, and possibly a description of the more important reasons why the outcome was what it was. RECORDs will consist either of lists of descriptive features or of pointers to other RECORDs. The information in a RECORD should not be constituent structure (i.e. SLOTs or NODEs) for the simple reason that one expects the RECORD to be remembered considerably longer than any constituent structure it would refer to.

Beyond these general observations, it is not clear, at this stage in the research, what specific facts will need to be recorded and what choice of properties would structure them in the best way. This is because the facts are both a function of what questions&discourse predicates&will need to be asked (something that is just beginning to be explored), and of the operations that could be used to go from the absolute positional information in the RECORDs to the relative positional information that the predicates will actually use. Discourse predicates invariably ask deictic questions: "has this event ever occured *within this paragraph*?" or "was the VP deleted in *the previous* conjunct?". At the moment there are too many degrees of freedom available to the designer to make a cogent decision on this matter; consequently, in MUMBLE many different designs are being experimented with. Below is the RECORD made by MUMBLE for the realization of the first instance of the message element macbeth in the example on page <macbeth_example_n.y.w.>.

[macbeth R1]

Position: clause index *c55* depth *1*
containing slot: *[subject]*
sentential context: *toplevel-nominal*

Realized as: *(proper-name)*
Strategies used: *(word-entry32 use-word)*

The information collected in RECORDs like this is never accessed directly. Instead, discourse PREDICATEs are written that compare the information in two RECORDs or between a RECORD and the current context, and then return a description of how the two are related. The most thorough example of this process appears in the section on subsequent reference (VII.B.1.1).

The same considerations apply to HISTORYs as apply to RECORDs. A HISTORY is a body of RECORDs about the same generic event, structured in such a way that one can recover properties such as the temporal order or recent instances, whether or not there was an instance in, say, the last paragraph, what role the event played the first time it occured in the discourse, and so on.

The proper structure for a HISTORY is even less clear at this stage than that of a RECORD. It is clear that the simplest structure, a sequence of RECORDs in chronological order, would be enormously inefficient to search once the discourse was longer than a few paragraphs. A compact, predicate oriented description is necessary if HISTORYs are to be reasonable objects to reason with.

4.1 Garbage collection and the compaction of the discourse history

Even if memory size were not limited, it would still be important to compact the information in the discourse history at regular intervals. The reason is very simple: as an event recedes into the past, its relevance to current decisions becomes smaller. At the same time, the

space of past events that discourse predicates have to search becomes larger, and a premium is placed on succinct description.

Constituent structure can be garbage collected (expunged) as soon as the controller has passed through it. That is, every time the controller leaves a NODE,[6] that NODE, its immediate constituent SLOTs, their contents, and any other temporary objects below them in the tree can be expunged from the system—the storage space they occupy is returned to the system free storage list and the objects cease to be defined.

The discourse history cannot be garbage collected so glibly. Past events should not disappear altogether, but should fade slowly. Each addition of a new RECORD to a HISTORY should cause the rest of it to contract, e.g. when a paragraph ends, the individual RECORDs for each of its sentences should be expunged and replaced functionally by properties on the RECORD for the paragraph.

5. The morphology routine

All words that originate in constituent SLOTs pass through the morphology routine before they are printed. The routine receives a WORD, examines its pname and its position in the tree, and produces a pname with the appropriate morphographemic shape which is then immediately sent to the output stream for printout.

As the name suggests, the morphology routine is the part of the linguistic component that knows the mophological facts of English. In MUMBLE, this knowledge is not extensive. The routine knows the productive verb forms, plurals, possessives, and contractions with *not*, some adjustments to determiners (e.g. "*a*"+"*other*" -> "*another*") and the structure of the verbal auxiliaries ("aux-hopping"). Its knowledge of morphological adjustment rules is limited to such things as "change the *y* to *i* and add *es*" and the doubling of final consonants after stressed vowels (marked as such by a property on the WORD). It does not know anything about creating new words from old via semi-productive morphemes such as *un-* or *-ment*.

The morphology routine is the only part of the linguistic component that deals with pnames, i.e. the only part to employ a sequential, character string based representation. By contrast, the message level is a conceptually-based network structured by the enumeration function, and the extended surface structure level is a (heavily annotated) syntactic tree accessed by deitic variables. This is because it is only in this last stage of the production process—just prior

6. As a convenience for debugging, MUMBLE typically waits until entire sentences are left before garbage collecting them. This is a load-time parameter.

to actually printing the text—that character string information is needed. Before that time, it is easier to reason in terms of a WORD's name and the individual features of the linguistic context, especially SLOT-NAMES. (Even a poetry program would find it more convenient to manipulate a *schematic* representation of sylable structure and rhyme patterns than to work with the pnames directly, if only because that avoids needing to know how to spell!)

A simple transducer

The ENTRYs select WORDs and position them in a grammatical frame. The morphology routine receives these words, one at a time, along with a description of their position. In MUMBLE, the description is conveyed by grammar-variables, especially *current-slot*, and *context-of-the-proposition*, and is used for determining how the first word of the verb group (the "tense bearer") should be treated. On the basis of that information (and a small amount of state information, see below), it then determines what pname to have printed. It is effectively one large conditional procedure, e.g. one of its cases willbe: "if the *current-slot* is [head] and the *current-NP* has the hook *plural*, compute or lookup the correct plural form for the word and use it instead of the regular pname.

Representing 'the last thing said' The morphology routine does not pass pnames through to the printer as soon as they are received and processed. Rather, it maintains a "buffer", presently one pname long, which it uses in testing for pname—pname interactions. In MUMBLE, only certain WORD classes, marked by a feature, are actually buffered. This is done for economy reasons since the grammatical phenomena that the buffer is used to test for effect only a few word classes. In a speech-based system however, the buffer would be in constant use because of the need to stage contextually induced allophonic variations.

Because of this buffer, the morphology routine is the most convenient place to define the relation "the last word said", that plays such a critical function in the grammar of the English auxiliary. The buffer is also used for contractions and adjustments to determiners (i.e. "*a + other => another*", "*any + <person> => anyone*"). Unlike the auxiliary, these actions are matters of free choice, not requirements of English grammar, and as a consequence, the morphology routine applies full-fledged GRAMMATICAL-DECISIONS to decide whether or not to make the combinations.

State information from the controller As discussed earlier in section <flags_for_grammatical_events_n.y.w.>, the morphology routine must be sensitive to certain events that are defined by the passage of the controller across certain syntactic "landmarks" rather than by the reception of certain WORDs. For MUMBLE, these are: the end of the [determiner] (to mark the possessive), the beginning of the [subject] (for inverted auxiliaries), the first word of the verb phrase (for tense, negation, and initial adverbs), and the end of one sentence—beginning of the next (for final punctuation and captializing the first word of the next