REL: A RAPIDLY EXTENSIBLE LANGUAGE SYSTEM

F. B. Thompson, P. C. Lockemann, B. Dostert and R. S. Deverill
California Institute of Technology, Pasadena, California

What is REL?

REL is an integrated information system designed to facilitate conversational access to a computer. As such it is similar in gross respects to many current developments in time-shared, conversational computer systems. However, in its system architecture it differs from other systems in:

a) a single language processor which accommodates a variety of user languages;

b) integrity of user data structures, user language and user data base;

c) tight coupling between the multiprogramming operating system and the single language processor;

d) rapid, conversational extensibility of user languages at the language processor/operating system level.

In the first two sections of this paper we review the design philosophy which gives rise to these features, and sketch the system architecture which reflects them. Within this framework, we have sought to provide languages which are natural for typical users. The third section of this paper outlines one such application language, REL English.

The REL system has been implemented at the California Institute of Technology, and will be the conversational system for the Caltech campus this fall. The system hardware consists of an IBM 360/50 computer with 256K bytes of core, a drum, IBM 2314 disks, an IBM 2250 display, 62 IBM 2741 typewriter consoles distributed around the campus, and neighboring colleges. Base languages provided are CITRAN (similar to RAND's JOSS), and REL English. A basic statistical package and a graphics package are also available for building special purpose languages around specific courses and user requirements.

Several such special applications are already being prepared: for example, a "theorem prover" based upon the lower predicate calculus and the resolution principle, an animated motion picture capability, and a syntax analyzer which allows grammars to be built from the typewriter and immediately applied to sentence analysis. The most extensive application so far is to a data base gathered by Caltech anthropologist Dr. Thayer Scudder. The data consists of over 100,000 items concerning the Tonga, a people living in Zambia who have made very dramatic cultural advance during the last decade. Dr. Scudder is already beginning the analysis of his data using his extension of REL English.

Underlying Design Considerations

The notion of user oriented language and of a user's data base are current notions in computer science. We should like to call attention to a particular relationship between them. Data, by its very nature is usually at a rather low level of aggregation.

```
What was the price of CDC stock in January
of the last three years?
 44.50 in January 1967
116.00 in January 1968
140.50 in January 1969
```

However, the user, in investigating a large body of such basic data, soon is concerned with notions that require a variety of aggregations of this data.

```
What were the averages between 1967 and
now of SDS, RCA and Digital Equipment
Corporation stock prices?
SDS                            105.30
RCA                             49.80
Digital Equipment Corp.        115.72
```

Some of these aggregations may arise as natural outgrowths of his analysis, and recur in his ongoing considerations.

```
What is the number of manufacturers of com-
puters whose memory size is greater than
32000 words and add time is less than 2
microseconds?
9
```

In order to maintain a reasonable efficiency of expression, he would like to define them once and for all.

```
def: large computer: computer whose
memory size is greater than 32000 words
and add time is less than 2 microseconds.
Defined.

def: power coefficient: high speed memory
size/add time.
Defined.

What computer manufacturers make large
computers whose power coefficient is
greater than 400?
Burroughs
Control Data
General Electric
Honeywell
IBM
```

This ability to extend his language naturally and easily is an essential system requirement.

These definitions are not merely more complex reformulations of the original language, independent of the data base. Nor are they simple reclassifications of the data. They can serve both to extend the syntactic aspects of the language and to form new relations and categories in the data. The excessive ambiguity found in the purely syntactic analysis of natural language sentences is reduced by contextual restriction of notions defined at higher levels of abstraction. Of course, if a query is ambiguous in terms of the given data base, the analysis should preserve it:

What were the yearend total installations of computer manufacturers whose stock prices exceeded 200 in 1967?
Ambiguous answers:

(1)	IBM	36000 in 1967
		42100 in 1968
(2)	IBM	36000
	Burroughs	1310

The query may be disambiguated by reformulation:

What were the 1967 yearend total installations of computer manufacturers whose stock prices exceeded 200?

IBM	36000
Burroughs	1310

What were the yearend total installations of computer manufacturers whose 1967 stock prices exceeded 200?

IBM	36000 in 1967
	42100 in 1968

The data and the language must be a highly integrated, user-oriented and extensible package. It is this inextricable nature of data and language that we want to stress here. In the early days of computers, this was one of the great ideas, credit for which was given von Neumann as the inventor of the "stored program computer." Today, with large operating and language systems that are independent of the user, we may have lost sight of this integral relationship. We may feel that at least the base level language can be user independent. Notice, however, that in the examples above, time is interpreted in units of one month. In a language/data package for a travel agent, time would be in minutes and ignore any reference to year; a neural biologist would measure events in microseconds, a geologist in millenia. Consider as another example the meanings of "where" in an anthropological investigation, in a picture processing language, and a police data file. Thus primary goals of our REL project are to facilitate the implementation and subsequent user extension and modification of highly idiosyncratic language/data base packages. In such a package, the semantics of the language can be specifically oriented in the context of the associated data.

Probably the most idiosyncratic aspect of any language/data package, other than vocabulary and definitions, are its data structures. If the package is to function efficiently in terms of response times, the structures into which the data is organized must reflect the conceptual organization in terms of which the user visualizes his data. It is this reciprocal relationship between language structure and data structure that underlies the effective use of a language as a tool for understanding.

From this reciprocal relationship we can derive two key implications for the architecture of the REL system. First, the system cannot preclude some forms of data structures that may occur in language/data packages. Thus one user may work with his data as contiguously stored arrays, another as n-plexes in a ring structure, a third may wish to manipulate strings of symbols. Second, the system can exploit the tight coupling between the syntactic processing of the sentence and the semantic processing of the underlying data structures. The more both the language and the internal data structures reflect the user's conceptual structures, the tighter this coupling will be. Although the forms for data organization are unrestricted, it can be assumed that syntactically distinguishable entities and data base entities will be reciprocally related.

One of the major problems in designing an operating system is peripheral storage management. If the language/data packages to be implemented involve large bodies of data or extensive calculation and language extension, peripheral storage management becomes a central concern. The clue to its solution is found in the interrelation between syntactic structure and data structure emphasized above. Consider the two general schemas of system architecture in Figure 1. In configuration B, user languages are defined in a manner independent of the language processor, in terms of their own grammar and associated semantic routines that handle their idiosyncratic data structures. The operating system, with its peripheral memory handling, is closely coordinated with the language processor. Thereby the dynamic allocation, and subsequent input/output, of peripheral storage is coordinated with syntactic analysis. In turn, the syntactically meaningful elements of sentences are recognized and processed by the language processor, thus carrying through the coupling between the language level and the storage management level. What the language's semantic routines do with the elements of memory made available to them is left up to the language designer, and not precluded by either the language processor or the operating system.

If on the other hand, as in system configuration A, the operating system and language processor are independent, the operating system's management of the peripheral storage will not be sensitive to the interrelationship between language and data structures, and the data will soon be spewed haphazardly over peripheral memory, with degradation of system performance as a result. One can prevent this by dictating structural aspects of the data, by establishing formating conventions; but this solution, as we have seen, will often prohibit an adequate representation of the aforementioned interrelationship. For these reasons, a central feature of the REL system is that it employs a single language processor integrated with the operating system and showing special concern for management of peripheral memory.

One of the principal interests of the REL project has been language/data base packages where the language has extensive structural power of expression, in particular those found in natural languages. If varieties of such languages are to be accommodated, the single language processor must be a powerful one. In particular, it must handle most rewrite rule grammars and simple transformational grammars, and must seek all analyses of an input sentence. We are particularly pleased with the language processor we have developed, which satisfies these criteria, and yet remains fast and tight. Central to it is the parser designed by Martin Kay of the RAND Corporation. Besides general rewrite and transformational rules, the language processor must handle ambiguity, language extension, and a variety of generator functions.

Architecture of Language Processor and Operating System

Language Processor

The language processor is a syntax-directed interpreter. A language is defined to the system in terms of its

(1) terminal symbols and parts of speech,
(2) rules of grammar,
(3) semantic routines.

The terminal symbols recognized by the system are all of the printable characters and constitute the symbols used in communicating with the user. The parts of speech identify the syntactic categories of words and phrases and participate in the grammatical analysis of sentences. The language designer chooses them at will and expresses them to the system implicitly through the rules of his grammar.

A rule of grammar has two parts. The first part consists of a sequence of parts of speech and terminal symbols that identify the constituents of a grammatical string; for example, "<number> + <number>" (where the brackets < and > are used to set off parts of speech from terminal symbols). The second part indicates the appropriate categorization of the phrase as a whole. In the simplest, most common case, this is a single part of sneech, and the rule may be formulated as a context-free rewrite rule, for example:

<number> → <number> + <number> .

Because it is conventional to write grammar rules in this generative form, we shall refer to the first part of the rule, the one to the right of the arrow, as the "right hand side." Similarly, we call the part to the left of the arrow the "left hand side." A rule of grammar may be more complicated than shown in the above example. In a general rewrite rule, the left-hand side may itself be a sequence of parts of speech and terminal symbols. A transformational rule may invoke a complicated check of the syntactic environment to determine if the rule applies and what its resulting phrase structure will be. Thus in REL, the second part of the rule of grammar is declared to the system as a "syntax completion routine." In its simplest case, such as in the example, it merely indicates a single part of speech for the resulting phrase.

Finally, a semantic routine is associated with each rule of grammar. In essence, the semantic routine determines the meaning of the entire phrase from the respective meanings of its constituents. In our example, this routine carries out the addition.

In conclusion, an REL language is a set of rules. Each rule has three parts:

(1) a right hand side,
(2) a syntax completion routine,
(3) a semantic routine.

For each language, the right hand sides of the rules are built into its grammar table which takes the form of a tree structure. Each rule in the tree points to its syntax completion and semantic routines.

At the core of the language processor is the parser. As it proceeds, it builds a parsing graph which constitutes the developing syntactic analysis of the input sentence. As the parser scans this graph, it recognizes the applicability of rules on the basis of the grammar table. When it recognizes a rule, it calls the associated syntax completion routine which develops the output phrase or phrases. The parser in turn builds these into a parsing graph, and proceeds until all possible rules have been applied.

Each phrase in the parsing graph that has a meaning relative to the data base contains a pointer or pointers into the appropriate data structures. When a rule is seen to apply, it is an open question whether the associated semantic routine should be executed together with syntax completion, or whether the semantic routines should be compiled along the way and exercised only after syntactic analysis has been completed. Immediate application of the semantic routine often shows a given syntactic construction to be meaningless, and therefore goes far in controlling excessive syntactic ambiguity all along the way. On the other hand, semantic routines often are highly complex, and sentence analysis may require excessive time if many spurious parsings occur along the way, especially when dealing with large data bases. This question is thus left to the discretion of the language designer who can indicate his choice to the system.

When postponing the semantic analysis, the language processor must compile information during the syntactic analysis which enables it to subsequently perform the appropriate semantic routines. Since the syntactic analysis reflects the sequence in which the semantic routines are to be exercised, the compiled information consists, in principle, of a list of these routines in precisely the order in which their corresponding rules applied. The information also includes an argument structure which the semantic routines ultimately process into the meaning of the total sentence — into the answer in the case of a question, into appropriate changes in the data base in the case of a declarative sentence.

So far we have not spoken of extensions of the base language. However, the capability to postpone semantic processing through compilation of the semantic routines provides all the means necessary to handle language extension.

When extending a language, the user specifies a new rule by identifying (1) the newly defined term, and (2) the defining expression. A typical definition may be:

def: f(x, y): x*x + 2*y

where x and y have previously been declared by the user as variables. The syntax of the defining part, e. g. x*x + 2*y , will have been carried to completion, indicating in this particular case that the result is a number phrase. Thus an associated syntax completion routine can be built by the language processor. The semantic routines are automatically compiled during the analysis of the expression, together with the appropriate argument structure. This compiled information represents the semantic routine of the new rule. The definition also determines the right-hand side of the rule. In the example, it consists of the sequence of symbols and parts of speech: f(<number>, <number>), where variables have been replaced by the parts of speech they denote. Whenever this right-hand side is encountered during an analysis, its constituents replace the variables in the defining expression prior to execution of the semantic routine. Hence, those elements in the argument structure of this routine reflecting variables must identify the corresponding constituent, e. g. by its position in the right-hand side. In summary, the language processor allows a user to be completely unaware of the internal representation of his language, but is nevertheless in a position to derive all three ingredients of a new rule, right-hand side, syntax completion routine, and semantic routine.

When the language processor encounters the semantic routine associated with a rule that was obtained through extension, it calls the listed routines in the given order. Before passing control to each routine, the language processor must select its constituents from the argument structure. Upon return from a given routine, the language processor must account for the routine's output by adjusting the argument structure, then call the next routine. After the list of routines has been exhausted the result must be passed to the parser in the appropriate form. We say for short that the language processor "generates" the sequence of calls.

Handling of "generating processes" is one of the most important functions of the language processor. Definition expansion is one such generating process. The following can also be described as generating processes:

General rewrite rules: In a general rewrite rule grammar or a transformational grammar, there are rules that result in a sequence of several output phrases rather than a single one. In this case, both syntax completion and semantic routines consist of a sequence of elementary routines each of which must be exercised in turn.

Ambiguity: If a word or phrase is ambiguous, each of its ambiguous meanings must be considered individually when it participates in the subsequent application of a rule. The phrase resulting from the rule will, of course, remain ambiguous unless all but one of the presented ambiguous meanings are excluded by the context of the rule. Propagation of ambiguity may be extensive.

Numerical quantification: Central to all programming languages is the notion of a loop. In higher level languages, this often takes the form of a "do" or "for" statement, such as "compute f(x) for x = 1 to n."

General quantification: In ordinary language we have such expressions as "all" or "some." Thus in processing the sentence: "Does some computer have a power coefficient greater than 5?", one must consider each computer in turn until one is found which satisfies the condition, or all are checked negatively. Similar considerations apply to "what," "how many," "at least five," "each," etc.

The importance of such generation processes cannot be overemphasized in considering the data handling capabilities of a system. It is these processes that are the primary tools for passing to higher levels of abstraction, for dealing with larger aggregates and conceptual elements. The contextual character of general rewrite and transformational rules, the use of ambiguous expressions which are then disambiguated in context, the employment of such words as "what," "some" and "each," all provide capabilities to rise above the base level of individual items of data to the perception of overall patterns and significances.

These generation processes: definition expansion, general rewrite rules, ambiguity, numerical qüantification, and general quantification, are all controlled by the same underlying generation process handler which uses typical stacking techniques. Thus all of these generating processes can be intermingled in a completely recursive manner. This we believe to be one of the most important strengths of the REL system.

The general schema Figure 2 for the REL language processor summarizes our previous discussions.

Operating System

Once a user has logged on the REL system he is asked to specify an option. At this point he can create a new language/data base package (which we call a "version"), delete one of his versions, list his versions, etc. Usually, the user will already have created a version which he then simply enters.

The activities of the user in the time between logon and logoff, and their effects within the system are referred to as a "session." The REL operating system is a multiprogramming system handling a number of sessions concurrently in core memory. The operation of the system is explained in Figure 3. When a user inputs a sentence his session is placed in the core queue. Once the session can be accommodated in core, it is moved to the CPU queue, its grammar table is read into core, and work space is allocated. After it is given control of the CPU, the language processor begins to identify and apply the rules in the language. The syntax completion and semantic routines are kept on drum or disk, the data on disk. Thus, repeated access to peripheral storage is necessary during sentence analysis. As a consequence, a session cycles between the CPU, the device queues, and the CPU queue during sentence analysis. When sentence analysis has been completed, the session relinquishes its core space. Any output resulting from the analysis has previously been spooled into buffer regions on disk. Displaying it on the terminal typewriter, as well as reading a new sentence, is handled by one of the "pseudo-sessions" in the system. These are treated like other sessions and differ from them only in not having grammar tables, thus occupying practically no core space.

The REL operating system can be characterized as being (1) sentence-synchronized, (2) service-driven, (3) multiprogramming, and (4) page-oriented. Analysis of a single sentence usually requires only a short amount of time, and at the end of the analysis little information must be carried over to the analysis of the next sentence within a session. A session resides in the central processor during the entire analysis. No time-consuming rolling of core space takes place, an advantage offsetting any delay that may occur when a sentence is not immediately processed because of lack of core space. Only modified disk records, and the grammar table if extended by a new rule, must be saved on completion of the analysis.

The high frequency with which transfers of information to and from peripheral storage take place allows the operating system to forego assumption of control at fixed-time intervals. The system gains control whenever language processor or language rules request such a transfer, or require some other system service.

Due to this large number of transfers, analysis of a sentence for a given session is interrupted frequently. As a consequence, utilization of the central processor is much improved if several sessions share it simultaneously. The maximum number of sessions which may occupy the central processor at a given time is determined by the number of grammar tables and work areas which can be concurrently accommodated. Whenever the operating system assumes control, it selects the session to be given control of the CPU on the basis of a priority scheme. This explains the two-way connection between CPU and CPU queue in Figure 3.

In order to prevent a session from monopolizing the CPU for an indefinitely long period of time, either because of extensive calculations or an error, analysis of a sentence is suspended whenever it consumed more than a given amount of CPU time. The session is placed at the bottom of the core queue, and the analysis later resumed. In this case, the entire core space occupied by the session must be rolled.

During analysis of a sentence, syntax completion and semantic routines and data are referred to in unknown sequence, some of them perhaps repeatedly. Therefore, peripheral storage is divided into memory pages of 1024 bytes each. The pages are distinguished with regard to their contents: "rule pages" contain routines, "data pages" the data bases. Accordingly the core space for holding pages is divided into two regions, one for each page type. A page-on-demand strategy is employed for space allocation within these regions. At any given moment, the rule page and data page last referenced by the session are locked into core. Pages and work space of a given session are protected against accidental destruction by other sessions.

Page organization of memory is also the means for achieving the tight coupling between the user's conceptualizations, language, and data structures. Each page can be symbolically referred to, and both language processor and semantic routines organize memory explicitly in terms of pages, according to their own specific criteria.

In order to aid in the design of the REL system, a detailed simulation model was constructed. The model was extensively applied and updated during design and implementation, using a variety of session mixes. We based both the nature of the mixes and the characteristics of the individual sessions on our experience with the existing Caltech time-sharing system, and an experimental version of REL in operation since spring 1968. On the basis of these studies we chose multiprogramming of four sessions, core space for 40 rule and 40 data pages, and a dynamically allocated work area of 64K bytes. For desk-calculator type questions the model predicts an average response time of 1.5 seconds for 30 active terminals, of 4.5 seconds for 50 active terminals. The corresponding response times for questions concerning large data bases, for example using REL English, are 15 and 25 seconds, respectively, and the average execution times for typical CITRAN (JOSS) programs 7 and 25 seconds.

REL English

Nature of REL English

In line with the design philosophy of REL, we have sought to provide as one of the base languages a language which usefully approximates natural English. In using the term "English" we bear in mind the fact that there really is no one English language. Rather, the term English refers to as many idiolects as there are speakers; these idiolects are grouped into dialects. The REL English is one such dialect. It shares with natural language also the characteristic of being, in its design and functioning, a conglomerate of idiolects, which we call versions. Such idiolectic versions arise when a user takes REL English as his base language, adds the vocabulary and relationships of his data, and proceeds to extend his language to fit his growing conceptual understanding.

The second basic characteristic is that REL English is a formal language. The characteristics of English as a formal language are discussed in an earlier paper [1]. The central thesis of that paper is that English becomes a formal language when the subject matter which it talks about is limited to material whose interrelationships are specifiabłe in a limited number of precisely structured categories. It is the type of structuration of the subject matter and not the nature of the subject matter itself that produces the necessary limitations. Natural language encompasses a multitude of formal languages, and it is the complexities of the memory structures on which natural language operates that account for the complexities, flexibility and richness of natural language.

How does our English compare with English as discussed by modern linguists? On the level of surface structure, they are essentially the same. Some more complex transformationally derived strings, such as certain forms of elipsis are not handled as yet. However, most of the common forms are treated in a straightforward manner. Although some constructions which can be formed in natural conversational English are not provided in the basic English package, such deficiencies can to a large extent be overcome by the capability for extension provided by the system.

The level of deep structure presents more problems. As distinct from surface structure, deep structure is that level of syntactic analysis which constitutes the input to semantic analysis. What is the nature of this semantic interpretation?

In the general case, little is known. In our case, as in most types of computer analysis, interpretation is in terms of the internal forms of organization of the data in memory. To the extent that the constituents of deep structure can be directly correlated with corresponding structures in the data, semantic analysis, and therefore sentence analysis, can be carried to completion.

It is important to distinguish, in this regard, between two quite distinct though related ways in which language use can be restricted. The first is by the ways in which the data is organized, that is, the structural forms used and the interlinkages which are formed for the manipulation of these structures. This type we will call "structural" restrictions. The second is by restrictions of the subject matter, the universe of discourse; this we will call "discourse" restrictions. When one restricts the universe of discourse to a body of material which occurs or is already derived in a formal way, one often tacitly accepts the structural restrictions thus imposed. To the uninitiated, it may appear that it is the discourse limitations and not the implied structural limitations that make the material amenable to machine analysis. However, it is the establishment of relatability between deep structural constituents and data structural forms, rather than discourse restrictions, that make computer processing of the semantic component possible. Any content area whose data is organized into these given structural forms can be equally efficiently processed by a system establishing such interrelationships.

The restrictions on REL English are a function of structural restrictions. Not all deep structures found in natural English are brought out by our analysis, because constituents of these deep structures do not correspond to structural relations in the organization of our data. For instance, "collections of boys" and "boys' collections" are considered synonymous. And yet, "At the fair, I saw collections of boys." and "At the fair, I saw boys' collections."

The REL English rules consist of a syntactic component, namely the right hand sides and the syntax completion routines, and the semantic component consisting of the semantic routines. The syntactic component constitutes a set of rewrite rules, context-free and general, which build the deep structure Phrase-markers in the form of kernel sentences, and a number of transformational rules. The semantic routines act on the memory structures of the data bases.

The parts of speech of REL English are given in Figure 4 together with examples. These parts of speech are inclusive terms for syntactic classes (labels on the parsing tree) and semantic categories (memory structures).

Function words, e. g. all, of, what, are distinct from referent words; the former are empty in the sense of not being associated with memory structures. Other aspects of the grammar, namely features, name and relation modification, verbs, clauses, quantifiers and conjunctions, are discussed in the remainder of the paper.

Ring structure of REL English Data Bases

REL English assumes that the data is organized in ring structures [2]. The semantic routines perform checks on, and modify these ring structures. Let us consider as our data base the following environment:

1) There are three ships, Maru, Pinta and Nina.
2) The respective locations of these ships are Boston, Boston and New York.
3) Boston, New York and Chicago are cities.
4) The respective populations of these cities are 3205, 11366 and 6689, in thousands.

Figures 5 and 6 illustrate how ring structures contain this environment. Let us follow these rings. Links from the SHIP ring pointing down and on the Maru, Pinta and Nina pointing up indicate the class-member relationship: Pinta is a member of the class SHIP. This exemplifies ring structures which relate classes of objects and individual objects. The other type of ring structure are binary relations between rings. If we follow the link from the Pinta ring through the location ring, we reach the Boston ring. The relation of location holds between Pinta and Boston, thus, in predicate notation, we can say: location (Pinta, Boston). The symbols 0, 1 and 2 refer to the order in this notation. The figures also illustrate the representation of numerical information.

The structural methods used to handle the time aspects of time-related data are not discussed in this paper for reasons of space.

Ring structures are also built during analysis of sentences and undone on their completion. Thus in processing the sentence:

What are the locations of ships?

the semantic routine for the "N → R of N" rule will build a scratch ring of all locations of ships, here Boston and New York. This scratch ring will be interrelated to the data base in the same manner as permanent rings and thus can participate in further analysis. After a given sentence has been processed, the scratch ring together with its links is removed from the data base.

Features

Features constitute an important and powerful mechanism in REL. We use the term in the general Chomsky sense [3], but with some significant modifications. Operationally, features can be thought of as flags which are checked and set by the syntax completion routines. Summarily, the role of features is subcategorization and order of syntactic groupings. Some typical examples of the function of features in REL English are the following:

(a) Morphemic features subcategorize parts of speech. These include the Nominative/Possessive and Singular/Plural subcategorizations of N and R; and the Singular and Participle subcategorizations of V. Morphemic features may also function as precedence markers for the order of syntactic groupings. Thus the Nom./Poss. and Sing./Plural features force the parsing:
"(((small (computer s))') (word size))", preventing "(small (((computer s)') (word size)))".

(b) The passive voice feature of verbs is set by the verb "to be" when used as an auxiliary. It is used in the processing of passive transformations. The negation feature of verbs is set by "not" and its contracted forms.

(c) "Precedence" features. These determine the applicability of rules, and the order of their application in forming syntactic groupings. On N and R they are:

(1) determiner - which prevents further modification (e. g. the the ball, big the ball, are excluded).

(2) phrasal - which prevents the application of certain morphemic rules to a syntactically modified phrase.

(3) pre-modification and post-modification - which cause the following constituent analysis:
e. g. (the (dentist (son of (Sally Smith)))) ,
not: (((the dentist) son) of (Sally Smith)) .
They also are used to maintain parallel construction in conjunctive phrases.
e. g. ((John's son) and (Mary's daughter))
not: John's (son and (Mary's daughter))
while: (Mary's (uncles and aunts))
not: ((Mary's uncles) and aunts)

For V-phrases, right modification and subject features play a similar role.

Precedence features are especially important because they sharply curtail redundant parsings. Thus the rules:

N → Ns N → NN
N → N's N → the N

give seven parsings of the phrase "the boy's dogs". Features eliminate all but one, namely: (the ((boy's) (dogs))) .

Name Modification

A "name" refers to a class or the individual members of a class in the ring-structured data base, e. g. "boy" or "John", a "relation" to a binary predicate. Name modification constructions consist of a name as head and one or more modifiers. Modifiers are: determiners, relations, and other names.

The indefinite and definite articles comprise the determiner class. Their functions are complex and an adequate discussion would go beyond the scope of this paper. Relations can function as pre-modifiers and post-modifiers. Accordingly, there are two different syntactic constructions, which we treat in REL as synonymous: "location of John" and "John's location." Figures 7 and 8 illustrate the application of such rules.

Relation Modification

We distinguish between two types of relations, primitive and complex. A primitive R directly designates an associated ring structure. A complex R designates a list structure consisting of constructions which build a complex relationship out of primitive ones. The process of building this list structure is referred to as "relation modification." Figure 9 shows the possible constructions, together with grammar rules for relation modification and examples of phrases to which they apply. Semantic routines for name modification recursively resolve this list structure. For example, in the analysis of the phrase "John's male child's location," "male child's location" gives rise to the complex R
composition [modification [male, child], location].
Then the name modification rule is applied to: "John's R," whose semantic transformation expands the complex R and develops the location of each male child of John.

Relation modification plays an important role in definitions of complex concepts. It is employed widely in the escalation of a language above a base level of primitive relationships. For instance, given the "parent," "male" and "identity" relations as primitive, the following series of definitions introduces the relation "uncle":

def:child:converse of parent
def:son:male child
def:sibling:child of parent and not identity
def:brother:male sibling
def:uncle:brother of parent

The Verb

A verb denotes an action or a state: an action is performed by an actor (subject) on an object, a state is momentarily or permanently frozen action between subjects and objects. Thus, an action, in this inclusive sense, is characterized by (1) the aspect of beginning, ending, duration or momentariness; (2) by its situation in time, and (3) by referring to subjects and objects. By virtue of involving subjects andobjects as well as temporal phenomena, the verb is focal to our linguistic expression. Two groups of verbs may be distinguished: those referring to a relation between subjects and objects, and those which establish a connection between them. The relation expressed by a verb constitutes its predicate; such verbs are called relation verbs. Relation verbs also express temporal aspects of a relation. Typical relation verbs are "arrive" and "leave"; both refer to the relation of 'location': "arrive" refers to the beginning of the existence of this relation, and "leave" to its ending. For instance, "John left Boston" means that the relation of 'location' existing between John and Boston came to an end. Verbs which express a connection between subjects and objects are referred to as copulas, e. g. "is" in "John is a boy." The copula itself constitutes the predicate.

1. Relation verbs. Each of the four following sentences contains, in surface structure, a different verb.

(i) John arrived in Boston.
(ii) John left Boston.
(iii) John lived in Boston.
(iv) John is residing in Boston.

The underlying structure of these sentences is identical: 'location (John, Boston)', except for the temporal aspect of this relation: beginning in (i), ending in (ii), momentariness in (iii) and duration in (iv). In REL English, the temporal aspect is denoted by the 'tense character' feature.

Verbs are introduced through language extension. They are defined in terms of a relation and temporal aspects. The relation must denote an already existing ring structure. Thus, the verb "arrive" is defined by:

def:arrive:verb(location, 1).

A verb is internally represented as a "verb table," with part of speech V. In the verb table for "arrive," 'location' is entered as predicate and '1' as tense character, indicating beginning. Initially, the verb table is assigned the present tense, or 'nowness.' This can be subsequently modifiedby verb morphology, auxiliary verbs, and tense modifiers (temporal phrases or clauses).

2. Copula verbs. The verb table for copula verbs has a copula predicate and no tense character. The copulas are: "is," "are," "was," "were," and their contracted negatives. For example, the rule 'C → wasn't' will result in the creation of a verb table with a copula predicate, time equal to past, and the feature of negation.

3. Verb table modification. The elements of a verb table correspond to the elements of a kernel clause (i. e., one with a single deep structure Phrase-marker). They are: subject, predicate, object, tense character, and times t_1, t_2. The subjects and objects of the kernel may be N (names) and U (numbers). The times t_1 and t_2 specify a single point or an interval of time in which the event indicated by the clause takes place. They may range from infinite past (0) to infinite future (∞).

REL verbs are marked by three regular inflectional morphemes: past tense, past participle, and 3rd person singular. The function of inflectional morphemes and auxiliary verbs is twofold: they modify the original time in the verb table and set the syntactic features. For example, the past tense morpheme and the auxiliary "did" modify t_1 to be past, thus establishing the time interval '0 to now' (i. e., $t_1 = 0$, $t_2 =$ now); the auxiliary "will" modifies t_2 to be future, thus establishing $t_1 =$ now, $t_2 = \infty$; 3rd person singular and auxiliaries "has,." "does" set the singular feature.

Time modifiers (M) further modify the verb table. They originate from prepositional time phrases, e. g. "before 1960," and subordinate clauses, e. g. "before John arrived in Boston." The times of the verb table are adjusted according to a given time modifier. Thus, in the sentence "Will John arrive before 1970?" the verb table for "arrive" has $t_1 = t_2 =$ now; modification by "will" changes t_2 to '∞'; modification by "before 1970" changes t_2 to '1970,' thus resulting in the intersected time interval 'between now and 1970.'

Subjects and objects are inserted in the verb table upon application of the rules:

(i) V → N V (rule putting subject on)
(ii) V → V N (rule putting object on)
(iii) V → V by N .

These rules also check whether the passive feature is on (set by copulas forming passives), and if it is, rule (i) converts the N into an object; rule (ii) converts the N into a subject; rule (iii) applies only if the passive feature is on and converts the N into a subject. Number subjects and objects do not participate in the passive transformation.

Negation is handled by a feature on the verb table. It is set by rules processing "not" with copulas, both in isolation and contracted, and rules processing "not" with verbs. They differ syntactically in the position of "not." If the negative feature is already set by a previous rule, the result is positive (i. e., double negatives turn into a positive).

Verb table accumulation is illustrated in Figure 10. The development of the verb table shows the successive modification due to the verb rules which have applied. The parsings reflect the use of features.

Clause Processing

The accumulated verb table constitutes the input to the clause processing routine. The verb table may be either complete, i. e., have all its positions filled, or else have subjects or objects missing. The clause processor completes the verb table if necessary, and returns it together with a list of times for which the clause holds, and another list of times for which it does not hold. The results of the clause processor are subsequently used by the sentence rules to produce output, or by the subordinate clause rules.

1. Sentences. Sentences are clauses plus sentence delimiters--in REL English, these are initial capital letters, and terminal question mark or exclamation mark. Accordingly, we distinguish two types of sentences, questions which interrogate the data, and commands which modify it.

The rules for handling questions are: (S stands for sentence, K for capital letter)

(i) S → K V?
Did Stan marry Jill before 1950?

(ii) S → K what/who V?
Who are the Smiths' children?

(iii) S → K when V?
When did John arrive?

Rule (i) results in a yes/no output depending on whether the relation holds at the specified time. Suppose the verb table is given as

predicate	spouse
subject	Stan
object	Jill
tense character	begin
t_1	0
t_2	1950

Suppose also the data show that Stan married Jill in 1948. Since 1948 is within the time interval '0 to 1950,' the answer will be affirmative. If the input is a negative question, we use the time for which the relation does not hold.

The output of rule (ii) is one or more Ns. These are supplied by the clause processing routine as either subjects or objects. In example (ii), subjects are supplied.

The output of rule (iii) is a time (or time list) at which the relation indicated in the verb table holds.

The output may be ambiguous. For example, "Did Smith live in New York?" would result in both "yes" and "no" if "Smith" referred to one Smith who did and another who did not live in New York. "Who is Stan's sister?" would have ambiguous answers if Stan had more than one sister.

The second type of sentences, commands, are used for adding or deleting data. The rules and corresponding examples are:

(i) S → K V!
John is a boy!
Sally lived in Boston after 1950!

(ii) S → K R of V!
The location of Sally was Boston after 1950!
The income of John is 10000!

The data is deleted if the verb table has the negative feature set. There are two restrictions on the above rules: the subject of the verb table in (i) must not be modified; and the predicate of the verb table in (ii) must be a copula.

2. Subordinate clauses. Subordinate clauses modify some item in another clause, or another clause as a whole. In REL English, clauses of the first type are always relative clauses; they are introduced by the pronouns "who," "which," "that," "whom" and "whose." Clauses modifying other clauses are temporal clauses introduced by "before," "after" and "when," and result in a time modifier.

Examples of relative clauses are:

(i) N → N who V
Did the boy who left Boston marry Jill?

(ii) N → N that V
Was Boston the city that John left?

(iii) N → N whose R V
Did the boy whose father left Boston marry Jill?

Verb tables representing relative clauses are completed using the noun which the clause modifies. Thus in (i) the N is used as the subject of the verb table, in (ii) either as subject or object. In (iii) the N for which the relation R holds is used as subject or object. Subsequently, the verb table is processed, in our examples resulting in a particular boy, or city.

Examples of temporal clauses are:

(i) M → before V
Did John marry Jill before Jill left Boston?

(ii) M → when V
What was John's income when Stan lived in Boston?

The output of rule (i) is a time modifier with the time interval $t_1 = 0$, t_2 = date specified by the subordinate clause. In the subsequent analysis these times are compared with the times in the main clause in the same way as time modifiers such as "before 1960." For instance, let us assume that John married Jill in 1950 and that Jill left Boston in 1955. The output of rule (i) will now be the interval '0 to 1955' and the times to be intersected are '0 to 1955' and 1950. The result, obviously, is a "yes" answer. The output of rule (ii) is a tense modifier with the time interval t_1 = t_2 = date specified by the subordinate clause.

Quantifiers

By "quantifiers" we refer to "all," "some," "how many," "what," "each" and "no." Phrases containing such words are of particular importance since they often are used in referring to aggregates and more abstract concepts.

Some examples are:

Did some Smith who lived in New York move to Boston?

How many Smiths have an income greater than 12000?

What was the average between 1950 and 1960 of the incomes of each Smith?

The language processor handles quantifiers through generation. In sentences with quantifiers a class of objects is considered. The sentence constitutes a condition on each element of this class, where the condition depends on the particular quantifier. Consider in detail the example:

"Is Boston the location of all men?" To answer this question, each man must be considered in turn. As each man is generated in turn, say man_i, the sentence "Is Boston the location of man_i?" must be processed. If the answer to all cases is "yes," then the original question is answered affirmatively; otherwise, it is answered negatively. In the case of the "some" generator, the answer would be affirmative if at least one of the men lives in Boston. For the question "What men live in Boston?", the answer is the list of men for whom the clause "man_i lives in Boston" holds.

Several generators may be nested within a sentence, as in "Each Smith lived in what cities?". In general, the nesting of generators poses difficult and challenging problems. Although we have been able to satisfactorily deal with these problems, an adequate treatment must be deferred to a separate publication because of their complexity.

Conjunctions

Conjunction of R's is treated as relation modification. Conjunction of N's and V's set up generators according to the following rules: $N_{GE} \rightarrow N\ JN$; $N_{GE}J \rightarrow N, NJ$; $V_{GE} \rightarrow V\ JN$; $V_{GE}J \rightarrow V, VJ$. The GE subscript indicates a generated phrase. If the J-phrase is "and," the generator is an "all" generator; if the J-phrase is "or," the generator is a "some" generator. An example is given in Figure 11.

Acknowledgment

The work reported in this paper was supported by grants from the National Institutes of Health and the Carnegie Corporation.

References

[1] Thompson, F. B., English for the Computer, Proc. AFIPS Fall Joint Comp. Conf., 29 (1966), 349-356.

[2] Craig, J. A., Berezner, S. C., Carney, H. C., Longyear, C. R., DEACON: Direct English Access and Control, Proc. AFIPS Fall Joint Comp. Conf., 29 (1966), 365-380.

[3] Chomsky, N., Aspects of the Theory of Syntax, MIT Press, 1965.

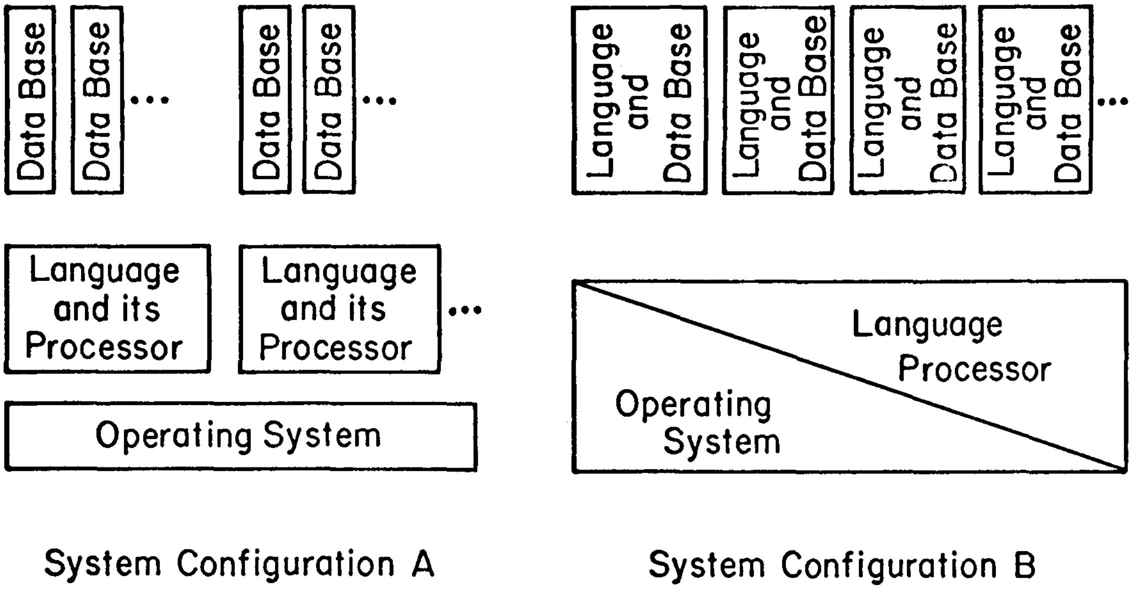


Figure 1. General Schemes of System Architecture

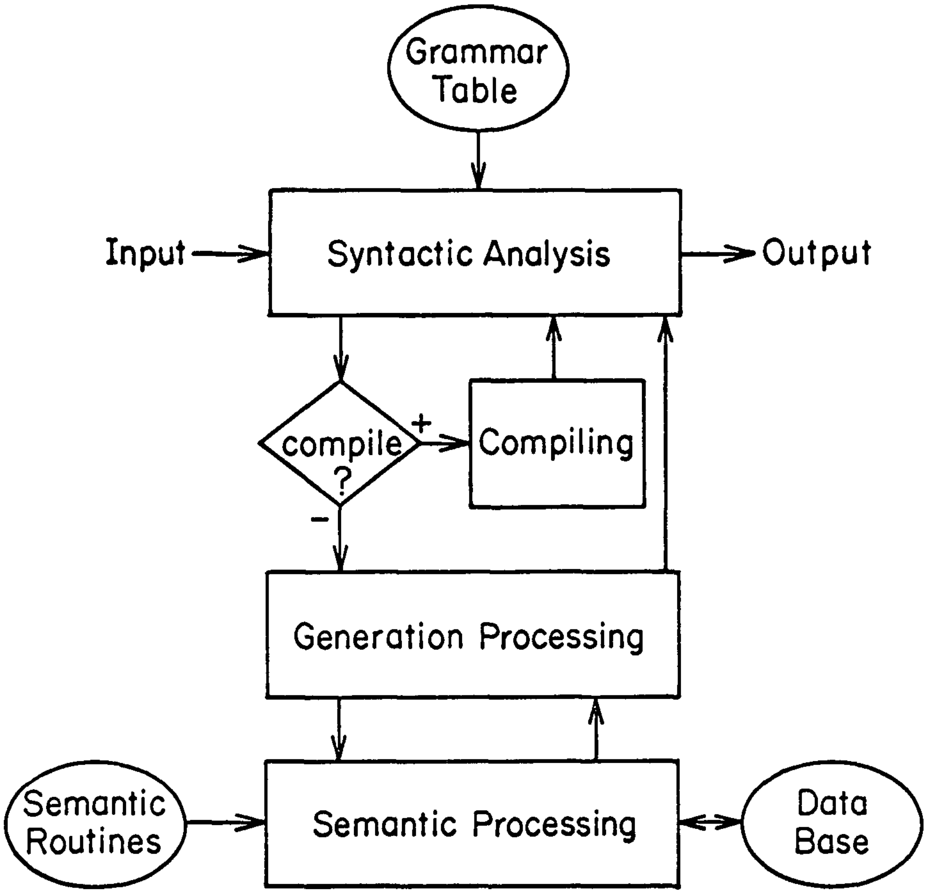


Figure 2. Language Processor Organization

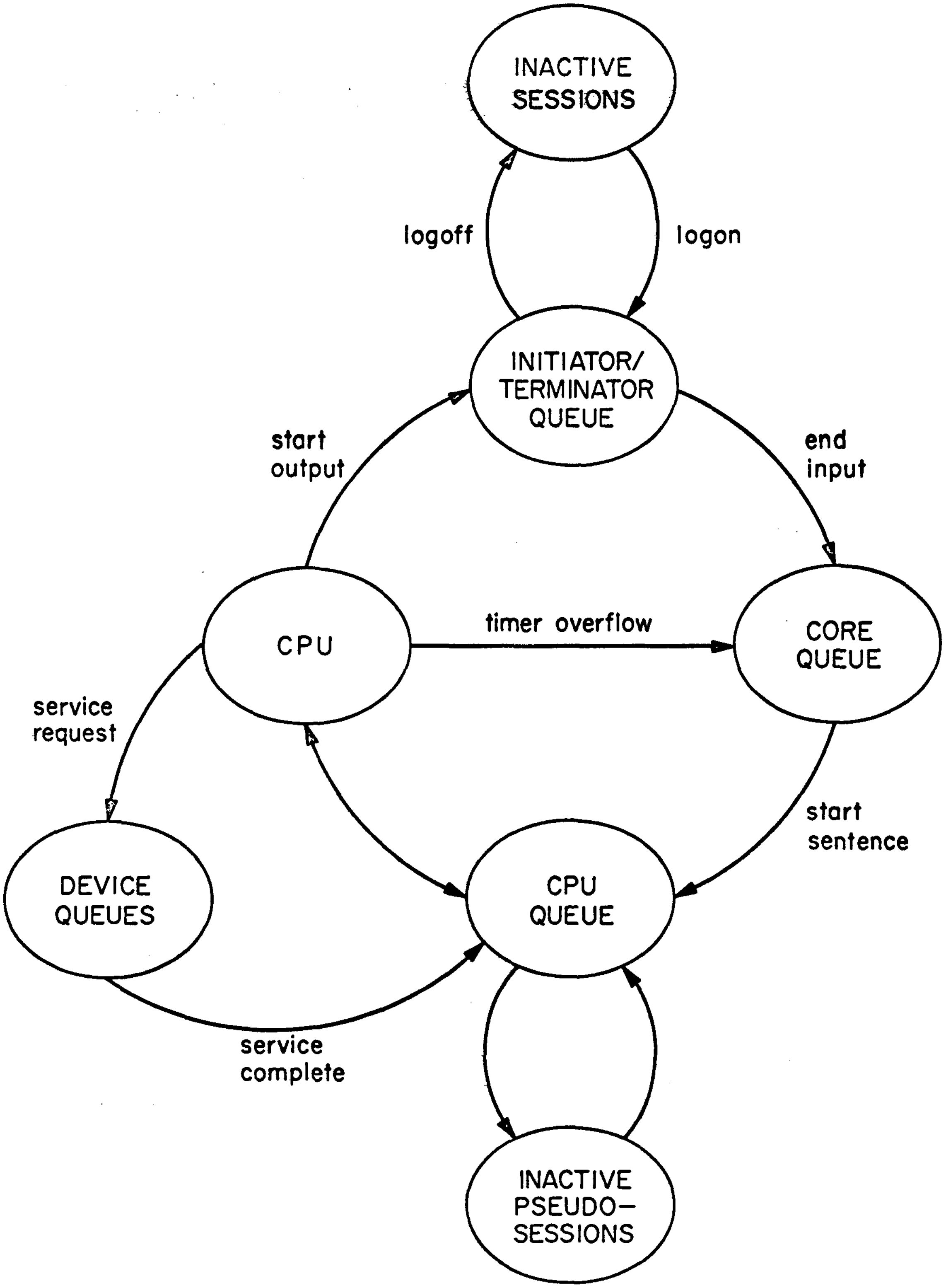


Figure 3. State Diagram for a Session

Symbol	Name	Examples
N	name	boy, John, the location of John
R	relation	location, father, age
V	verb table	arrive, has lived
C	copula	is, was
J	conjunction	and, or
T	time	January 1960
M	time modifier	before January 1960 between now and 1970
U	number	52, 61.34
S	sentence	John is a boy! Is John a boy?

Figure 4 Parts of Speech in REL English

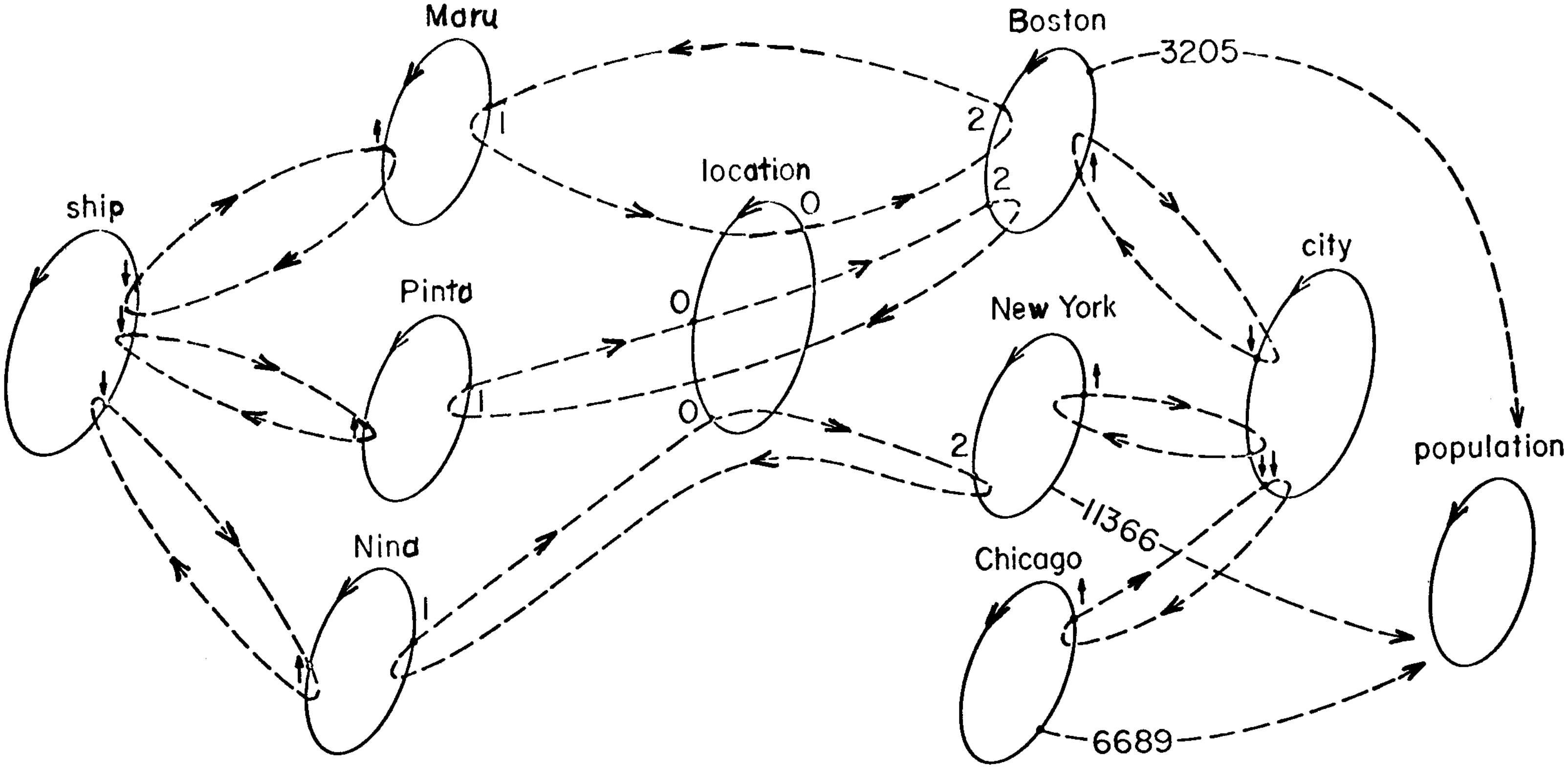


Figure 5. A Small Ring-Structured Data Base

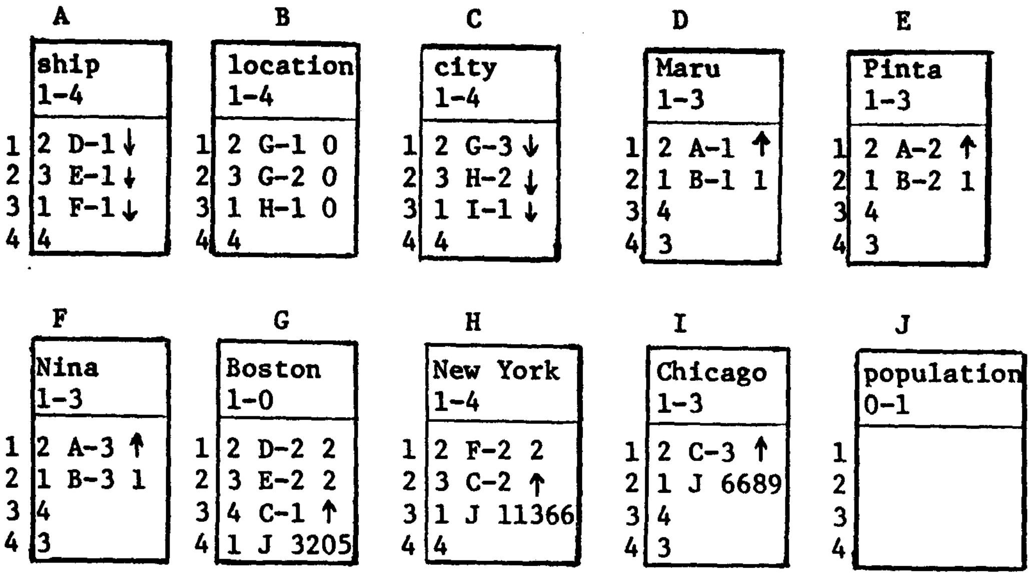


Figure 6 Schemantic Page Layout of Ring Structured Data

type of constituents		example	output N
R	N		
singular	singular	location of John	scratch ring of all locations of John, singular, phrasal, postmodified
plural	singular	locations of John	scratch ring of all locations of John, plural, phrasal, postmodified
singular	plural	location of boys	scratch ring of all locations common to all boys, singular, phrasal, postmodified
plural	plural	locations of boys	scratch ring of all locations of all boys, plural, phrasal, postmodified

Figure 7 Cases for rule N → R of N

type of constituents		example	output N
N (singular only)	N		
class	singular individual	scout Smith	Smith who is a scout if there is only one. List of such Smiths, marked as ambiguity, if there is more than one. singular, phrasal, premodified
class	singular class	boy scout	Scratch ring of all boy scouts. singular, phrasal, premodified
class	plural class	boy scouts	Scratch ring of all boy scouts. plural, phrasal, premodified
individual	singular class	Boston ship	For each relation that interrelates Boston with some ship, a scratch ring for all such ships Ambiguous over all such relations. singular, phrasal, premodified
individual	plural class	Boston ships	Same as above, but plural

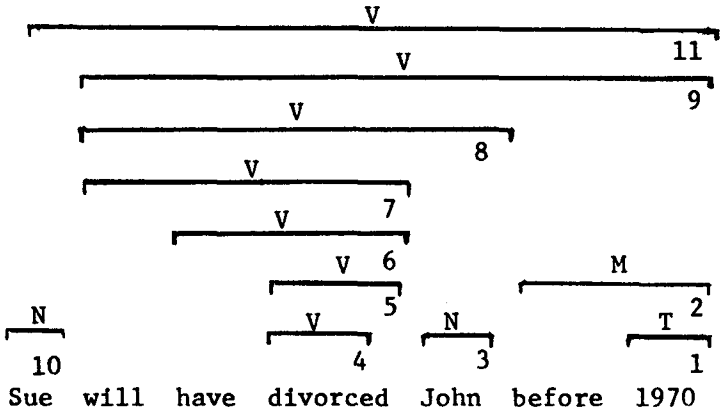
Figure 8 Cases of Rule N → NN

composition	R → RR	son's location
conjunction	R → RJR	uncles and aunts
disjunction		uncles or aunts
modification	R → NR	boy friend
relative negation	R → R but not R	uncles but not aunts
converse	R → converse of R	converse of parent

(for number relations)

sum	R → R + R	direct salaries plus overhead
difference	R → R - R	
product	R → R * R	pay rate * hours worked
quotient	R → R / R	GNP/population

Figure 9 Complex Relation Constructions



Verb Table	4	5	6	7	8	9	11
predicate	spouse	spouse	spouse	spouse	spouse	spouse	spouse
subject	-	-	-	-	-	-	Sue
object	-	-	-	-	John	John	John
tense character	end	end	end	end	end	end	end
t_1	now	0	0	0	0	0	0
t_2	now	now	now	∞	∞	1970	1970
participle feature	off	on	off	off	off	off	off

Figure 10 Verb Table Accumulation

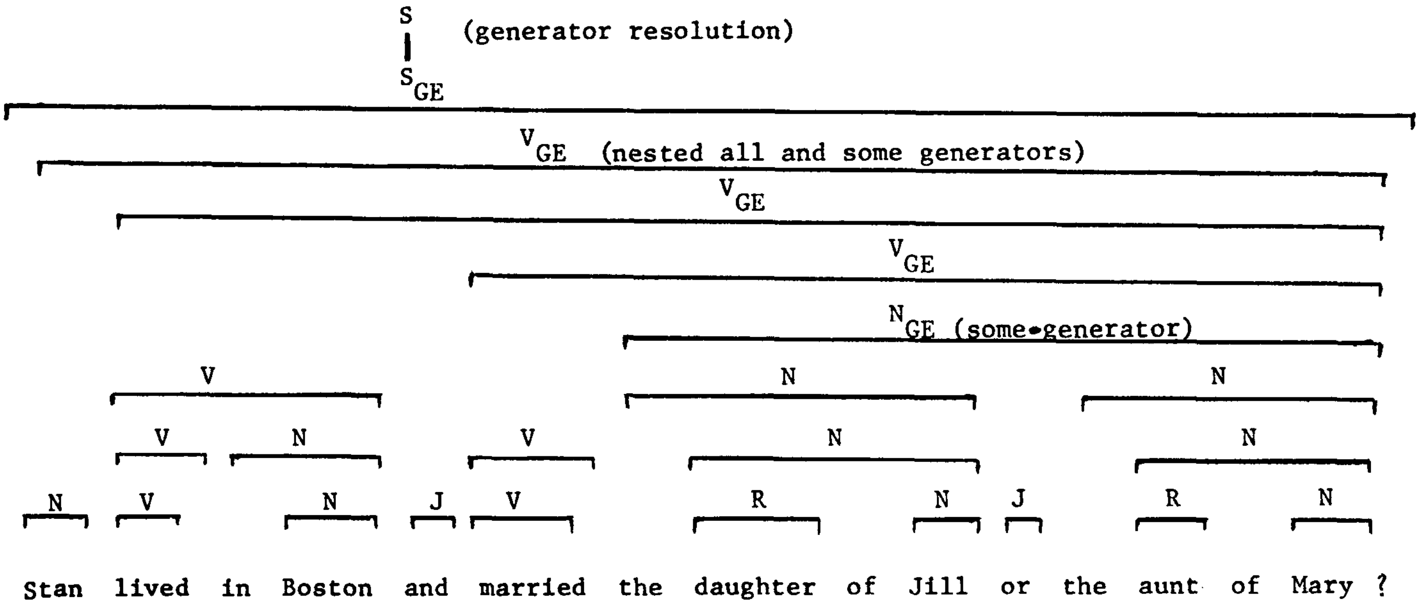
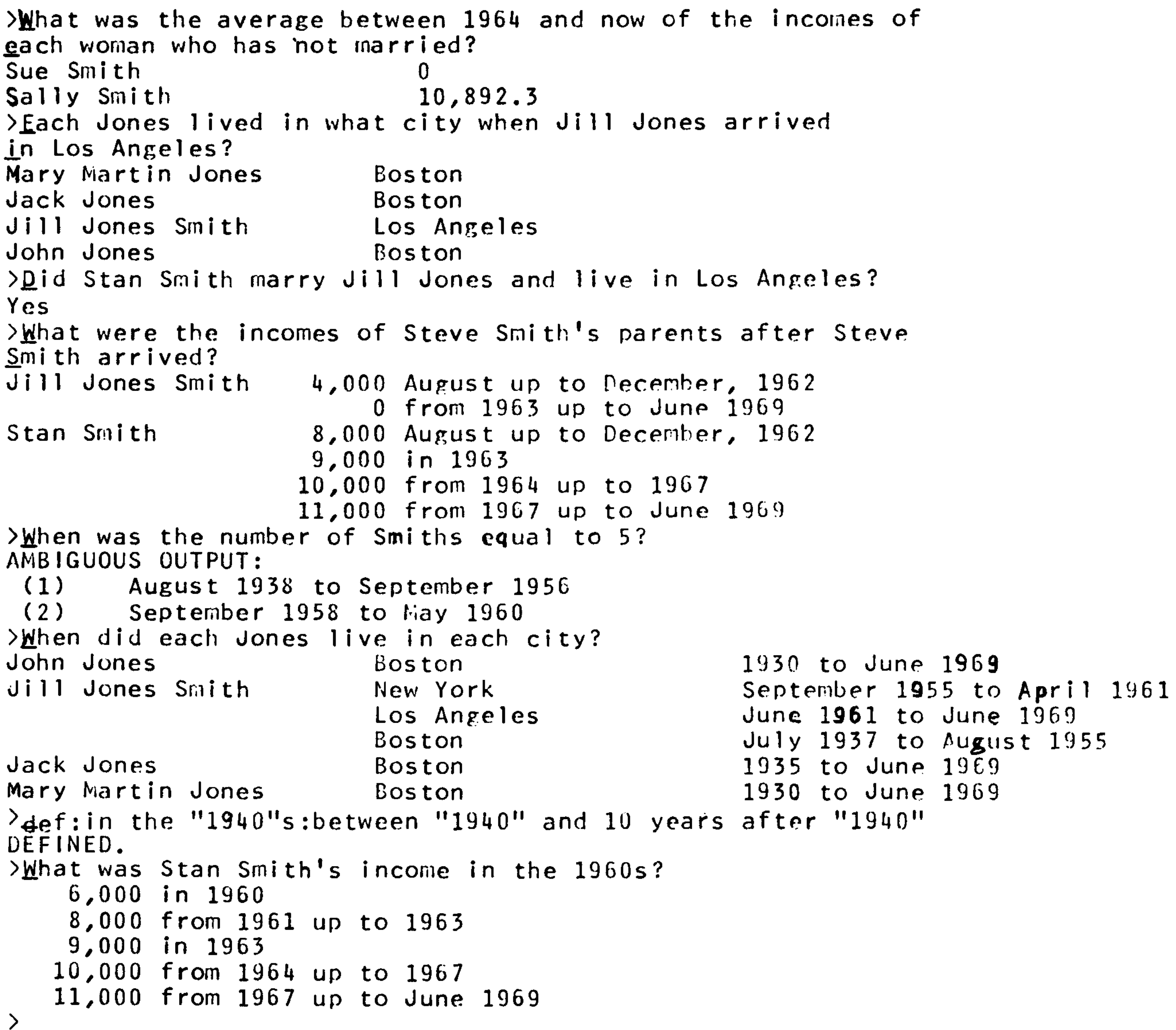

S
(generator resolution)
S_{GE}
V_{GE} (nested all and some generators)
V_{GE}
V_{GE}
N_{GE} (some•generator)
V
N
N
V
N
V
N
N
N
V
N
J
V
R
N
J
R
N
Stan lived in Boston and married the daughter of Jill or the aunt of Mary ?

Figure 11 Example of Conjunctions



```
>What was the average between 1964 and now of the incomes of
each woman who has not married?
Sue Smith                  0
Sally Smith                10,892.3
>Each Jones lived in what city when Jill Jones arrived
in Los Angeles?
Mary Martin Jones       Boston
Jack Jones              Boston
Jill Jones Smith        Los Angeles
John Jones              Boston
>Did Stan Smith marry Jill Jones and live in Los Angeles?
Yes
>What were the incomes of Steve Smith's parents after Steve
Smith arrived?
Jill Jones Smith     4,000 August up to December, 1962
                         0 from 1963 up to June 1969
Stan Smith           8,000 August up to December, 1962
                     9,000 in 1963
                    10,000 from 1964 up to 1967
                    11,000 from 1967 up to June 1969
>When was the number of Smiths equal to 5?
AMBIGUOUS OUTPUT:
 (1)    August 1938 to September 1956
 (2)    September 1958 to May 1960
>When did each Jones live in each city?
John Jones              Boston                  1930 to June 1969
Jill Jones Smith        New York                September 1955 to April 1961
                        Los Angeles             June 1961 to June 1969
                        Boston                  July 1937 to August 1955
Jack Jones              Boston                  1935 to June 1969
Mary Martin Jones       Boston                  1930 to June 1969
>def:in the "1940"s:between "1940" and 10 years after "1940"
DEFINED.
>What was Stan Smith's income in the 1960s?
    6,000 in 1960
    8,000 from 1961 up to 1963
    9,000 in 1963
   10,000 from 1964 up to 1967
   11,000 from 1967 up to June 1969
>_
```

Fig. 12. A Few Inclusive Examples