DATA BASE USER LANGUAGES FOR THE NON-PROGRAMMER

Peter C. Lockemann
Fakultaet fuer Informatik, Universitaet Karlsruhe
D-75 Karlsruhe 1

Abstract

In light of the necessary investments, commercially available data base systems usually offer comparatively general-purpose interfaces. These are suitable only for the data base specialist. In order for a data base system to attract non-programmer users, interfaces must be provided that approximate the special user terminology and conceptualizations. If, in particular, these users form a heterogeneous group, a variety of interfaces will be required. Questions of interest are then the extent to which user interfaces should be standardized, the techniques which allow rapid implementation of new more specialized interfaces, or the procedure for selecting the most suitable interface for a given problem. Based on the concept of hierarchy of abstract machines, the paper presents a possible approach to the solution of these questions. Three examples will be introduced to critically examine the concept and demonstrate some of its merits and shortcomings.

1 Introduction

The success or failure of a data base system, no matter how well-conceived it may appear to the author's mind, is ultimately decided by the users the system is supposed to serve. This aspect is often overlooked by system planners who devote almost their entire effort towards organizational problems such as analyzing the informational needs of an institution or organization, the current status of information flow within the organization and the necessary improvements to it. From the analysis a number of requirements are derived such as the extent of information integration, time characteristics, information system structure, adaptation of the organizational structure, relinquishment of old resources and provision of new ones. All too often, much less attention is being paid to the individuals who must use the system. They are simply expected to appreciate the needs of the organization and to adapt most willingly to the new environment.

Human nature, however, is conservative. Human individuals will cling to the same terminology and methodology and try to solve the same problems unless and until one can make a most convincing point for reorientation. In many cases data base systems are not even introduced to solve new kinds of problems. Rather they are supposed to improve the solution to existing and already well-understood problems, or at least use these problems as a point of departure. Under these circumstances there is no reason why users should be burdened with radical changes in style.

Unfortunately, for the manufacturer of a data base system this is just one side of a coin. For him, the development and implementation of a data base system represents a large investment which he can only justify by corresponding sales figures. This precludes him from attending to each of a large variety of individual user needs but compels him to offer general-purpose interfaces. On the other hand it is these general-purpose interfaces that prove repugnant to many a potential user who has his own special terminology, conceptualizations and application problems.

In order to resolve the dilemma, techniques must be developed that permit the adaptation of a data base system to various user needs. In particular, the solutions should address themselves to the following questions.

(i) How can user language interfaces be separated from the operational and management characteristics of the data base system?

(ii) Are there any techniques that allow, in a systematic way, for the rapid implementation of a user language according to given specifications?

(iii) To which extent is it economically feasible to construct and stockpile "off-the-shelf" user languages?

(iv) Given a set of language specifications, under which conditions can one build upon an already existing user language? Can one define a relation on user languages that formalizes these conditions and determines the amount of effort required?

To answer these questions we shall define a hierarchical relationship between user languages. The nature of the relationship will be discussed in some detail. A number of examples will be introduced to explicate the approach and to point out its merits as well as some of its present shortcomings. The discussion is intended basically for non-procedural interactive languages.

2 Hierarchies of user languages

2.1 Concepts

The hierarchy of language interfaces shall be defined as follows [Kr 75]:

- Each interface is defined in terms of a ("lower") interface, and may itself serve as the basis for definition of a ("higher") interface.
- There is exactly one interface which cannot be defined in terms of another interface and hence serves as the ultimate basis for all other interfaces.

Such a hierarchy of interfaces may be graphically represented in the form of a tree where each node corresponds to a particular interface.

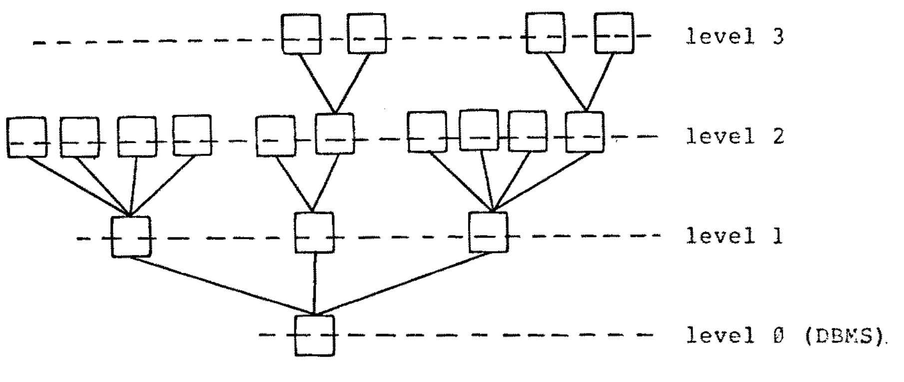


figure 1

The hierarchy must be chosen such that it reflects a hierarchy of users. Level 0 corresponds to the data base specialist, while level 3 might cater to a user completely untrained in computer affairs.

The previous questions can now be restated with a little bit more precision.

(i) Can all fundamental operational and management functions be solved underneath the basis on level 0?

(ii) What are the formal criteria that allow to construct a hierarchy by defining new languages in terms of existing ones?

(iii) Up to which level in the tree should interfaces be standardized?

(iv) Suppose a given language specification is represented as a node. Can a path to an existing node be constructed, and the length of the path be "measured"? Can one determine the path with minimum length? If the path is too long, should intermediate nodes be introduced, and what would be their specifications?

At this point in time, "length" is no more than an intuitive notion for which a formal measure does not exist. However, a rough outline of the definition of one node in terms of another one may often give some insight into the amount of effort necessary and thus provide an estimate of the length.

Language hierarchies have long been mentioned in connection with programming languages, e.g. Assembler - Low-level programming languages (e.g. PL 360 [Wi 68], ESPOL [Bu 72]) - High-level programming languages - Very high level languages (e.g., set oriented languages [SI 74]). However, except for macro languages these do rarely conform to the strict definition given above (e.g. COBOL is not defined in terms of a lower-level programming language), the reason being that this would entail inefficient compilation. The same argument does not hold for data base languages where language analysis is but a minor part of query processing [Kr 75].

2.2 Explications

The notion of hierarchy as introduced above is still vague and should be made more precise. Below several concepts known from the literature are introduced. Their usefulness as well as some of their deficiencies will be discussed in the remainder of the paper.

(1) Characteristics of the root.

There exist several schools that claim to provide the just and only basis for data base concepts. Before one may pass any judgment on these claims one ought to agree on the criteria that a basis would have to meet. It is commonly accepted that a data base is to be considered as the model of a certain reality. Hence a basis should be such that it provides concepts so primitive that any reality, be it physical or conceptual, could be adequately covered by it. Some authors [Ab 74, Su 74] have attempted to enumerate certain primitives: elementary objects, properties, relations, orderings, categories (or types), names, as well as sets of operators for creating, accessing, manipulating and deleting these. In addition, one might consider organizational questions such as parallelism and sharing of models by various users.

(2) Dependencies between successive nodes.

Since it is extremely general, the root is of little practical value to the average user. Users are invariably concerned not with all possible realities but with certain classes of realities, and wish their models to reflect the corresponding limitations. In other words, the modeling tools on level 1 will differ from those on level Ø by defining certain restrictions on the way the primitives may interact. The same obviously is true for level 2 vis-a-vis level 1, etc. These restrictions relate mainly to the manner in which objects may be composed into new objects, relations into new relations, and/or operations into new operations.

(3) Characterization of a node as an abstract machine.

Basically, the restrictions defined on the permissible compositions determine the dependencies between successive nodes. To make this a little bit more precise, the concept of abstract machine is introduced. An abstract machine is a set of object types, a set of operators for manipulating objects and defined on object types, together with a control mechanism that allows to construct and execute sequences of operations. Each node is then described in terms of an abstract machine.

(4) Dependencies between abstract machines.

By assigning an abstract machine to each node, the following properties must hold between two successive nodes A_i and A_{i+1} [Go 73]:

a) The resources and the functions provided by A_i form the complete basis on which to build A_{i+1}. There is no way to use properties of A_{i-1} in building A_{i+1}. Hence every A_i is a complete interface description in the hierarchy.

b) Resources of A_i used in defining new resources of A_{i+1} can no longer be present in A_{i+1} (i.e. they may become resources of A_{i+1} only if they are not part of a definition for another resource of A_{i+1}).

Keeping these rules in mind I shall attempt, as a matter of illustration, a tentative classification of some results discussed in the literature [Ab 74, Co 70, We 74, Kr 75, Wo 68, Wo 73, Gr 69, Col 68].

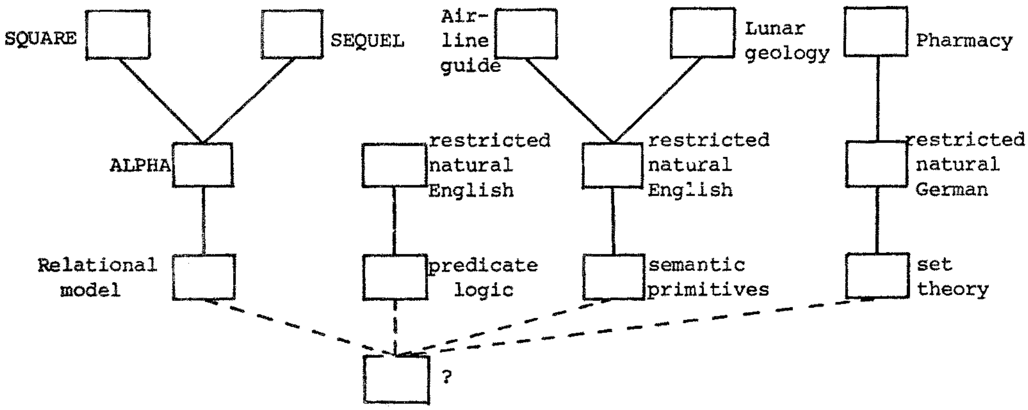


figure 2

2.3 Consequences

The concepts and rules introduced above impose a certain discipline on the design of user languages, on their application, and on the transition between them. Some of the consequences are outlined below.

(1) If we strictly keep to the rules above, a new interface must be defined in terms of its immediate predecessor and not any arbitrarily chosen predecessor, i.e. immediate predecessors must not be bypassed ("stepwise abstraction"). On the other hand, given certain specifications and a suitable node in a tree, intermediate nodes that hopefully are of general usefulness should be introduced on the intervening path whenever the path proves too "long" ("stepwise refinement").

(2) Given a path to the root, a user should be put into position - at least in principle - to formulate his requests in any of the languages that correspond to the nodes on the path. As a matter of fact, we found this an essential prerequisite for efficient system testing since system activities may be observed and controlled to any desired level of detail [Kr 75].

(3) Queries are stated on some level and must successively be translated between levels until the root has been reached. Definition (of an abstract machine) and translation reciprocate each other: The definition of the next higher level from a given one determines the rules that govern the translation of statements on the higher level to those on the lower level.

(4) Results are produced on the lowest level but must be presented to the user on a higher level. As a consequence, following the evaluation of a query a second ("reverse") translation must be invoked in order to propagate the results to higher levels.

3 Set theoretic basis

3.1 Motivation

The rules of ch.2 have been applied to the construction of the KAIFAS question-answering system and have proven highly useful there. Hence this system will be chosen as the first example to demonstrate the practicability of the rules. For a more detailed description of the system the reader is referred to the literature [Kr 75].

Restrictions with regard to the general basis are motivated by the realities one wishes to consider. In the case of KAIFAS we presume that relations are exclusively of the property type (sets) or are binary relations and, more important, that objects are selected exclusively on the basis of given properties or relations which they meet or undergo, perhaps in logical combination. Indeed one can show that the set theoretic approach may be viewed as a generalization of the inverted file technique [Kr 75].

3.2 Set theoretic machine

Object types

I Elementary objects (individuals), e.g. Hans Maier, Bonn, Aspirin

M Sets, e.g. city, medication
List of individuals.

R Relations, e.g. father, contraindication
List of ordered pairs of individuals.

Z Numbers

D Measures, e.g. 2 years, 4 tablets/day
Ordered pairs (number, unit expression).

F Measure functions, e.g. age, dosage
Lists of ordered n-tuples whose last components are measures.

B Truth values

Operators

On retrieval the machine is supposed to function in the following way. Set, relation, and function names refer to objects in permanent storage. In order to manipulate the objects they must be transferred into unnamed registers of which an unlimited number is thought to exist. Hence all operations except for the load operations are register-to-register operations.

Load operators

Mw, ev, en, ef	Load a set, a relation (ev, en), and a measure function, respectively.

Set operators

MU:	MxM->M	Union
M∩:	MxM->M	Intersection
Km:	MxM->M	Relative complement $\{x \mid x \in M_1 \wedge x \notin M_2\}$
Kz:	M->Z	Cardinality

Binary relation operators

Ko:	R->R	Converse relation
Rb:	RxM->R	Restriction $\{(x,y) \mid (x,y) \in R \wedge x \in M\}$
Rp:	RxR->R	Product $\{(x,y) \mid \exists z:(x,z) \in R_1 \wedge (z,y) \in R_2\}$
R∪:	RxR->R	Union

Reduction of binary relations

Vo:	R->M	Domain $\{x \mid \exists y:(x,y) \in R\}$

Na: R->M Range {x|∃y:(y,x)eR}
Vg: RxI->M Individual domain {x|(x,I)eR}
Ng: RxI->M Individual range {x|(I,x)eR}
VgU: RxM->M Restricted domain {x|(x,y)eR∧yeM}

Reduction of measure functions

Fw: FxI->D (n=2)

Logical operators

e: IxM->B Test on set membership
c: MxM->B Test on set inclusion

In addition, the standard logical operators are available as well as the standard arithmetic and comparison operators for numbers and measures.

Control mechanism

Sequencing of operations

"Programs" for the set theoretic machine are expressed in a functional notation. Operations are performed from left to right and, for each nested argument, from inside out.

Example: A question such as "Are cities birthplaces of engineers?" would take the following form in the set theoretic machine

c(Mw(M_{city}), VgU(en($R_{birthplace}$), Mw($M_{engineer}$)))

Loops

Loops are introduced by the use of bounded quantifiers which have three arguments:

1) An expression resulting in a set of objects (range).
2) An expression for the condition resulting in a truth value (scope); it may be regarded as the loop body.
3) The name of a bound variable; each of its substitutions defines an invocation of the loop.

Important quantifiers are
AL: MxB ->B all, every
EI: MxB ->B some
DB: MxB ->M which

ZB: $M \times B \rightarrow Z$ how many
with the left-hand M the bounding set and the left-hand B the condition.

Examples:

DB (x,Mw(M_{city}), e(x,VgU(en($R_{birthplace}$),Mw($M_{engineer}$))))

with the meaning of "Which cities are birthplaces of engineers".

DB (x_1,
    Mw(M_{manuf}),
    ZB(x_2,
        Vg(en(R_{prod}),x_1),
        DB(x_3,
            Mw($M_{ailment}$),
            e(x_2, Vg(en(R_{medic}),x_3))))))

with the meaning of "How many products of which manufacturers are medications for which ailments?"

<u>Expressions in the data base</u>

Set membership of an arbitrary kind is expressed by including, in the representation of a set, arbitrary set expressions. Example (in German):

$M_{rezeptpflichtig}$	
$I_{Spasmocibalgin}$	
Vg(en($R_{Derivat}$), $I_{Oxazolidin}$)	①
$I_{Morphin}$	
Mw(M_{Opiate})	②
Mw($M_{Hypnotika}$)	
$I_{Methadon}$	
Vg(en($R_{Derivat}$), $I_{Succinimid}$)	
Vg(en($R_{Heilmittel}$), $I_{Agitiertheit}$)	

where ① indicates all derivates of Oxazolidin to be prescription drugs, ② all opiates, etc.

This concept is extended to relations and measure functions. Two of its advantages are:

- Since all objects are evaluated on request only, changes to the data base may be made locally without regard to any interrelationships that may exist.

- Expressions may be stored without regard for the existence of any individuals for it. Hence one could construct a data base consisting exclusively of higher-order relationships.

One consequence, however, is that the control mechanism must itself be defined recursively since it may be invoked on any load operation.

3.3 Natural language

Few users will feel at ease with the highly stylized language introduced in sec. 3.2. One possible step of abstraction, therefore, is the definition of a new abstract machine accepting natural language input. By necessity this is a highly restricted form of natural language since its semantics, and hence its syntactic forms, can be no more than what may ultimately be reduced to a set theoretic interpretation. Moreover, it must be considered more restrictive than the set theoretic interface because while one may nest set theoretic expressions to an arbitrary depth, those beyond a certain depth simply cannot be stated in natural language in any comprehensible fashion.

To speak of objects, operators and control mechanism in connection with natural language turns out to be highly unnatural, or rather impossible. It is possible, however, to define an abstract machine on that level in terms of the syntax of the interface which in turn may still be based on object types. This is in striking similarity to Very High Level languages vis-a-vis High Level programming languages: Very High Level languages are loosely described as languages used to specify what is to be done, rather than how it is to be done [SI 74].

In accordance with sec.2.2, the object types must relate to the ones of the set theoretic machine. In this case the relationship is straightforward as indicated by the following list:

N proper names for the objects of the universe.
A attributes (properties of an object of the universe).
R references from one object of the universe to a second one (e.g. Thebacon is referred to by Morphium as its derivate).
M references to measures.
D numbers or measures.
S sentences. These are of two kinds: sentences to be answered by yes or no, and sentences to be answered by counting or enumerating proper names.

Some examples from KAIFAS in which German was chosen as natural language interface.

Ist Psyquil rezeptpflichtig?
N A
Betraegt die Tagesdosis von Chinidin 2 Gramm?
M N D
Welche Derivate von Morphium sind rezeptpflichtig?
R N A

The syntax of the interface is described by a grammar with the following general properties:

(1) Syntactical variables must relate to the object types, hence they cannot be based on the traditional grammatical categories such as noun, noun phrase, adjective, etc. but on categories that are essentially semantical in nature. The variables are IN(names), ME (attributes), RE(references), MF(references to measures), ZA (numbers), SA (sentences), QU (quantifiers).

(2) On the other hand, the traditional categories must be accounted for in some way, e.g. in order to reject incorrect inflections. As a consequence, each syntactical variable is indexed by a number of features. Examples:

MAS masculine)	NOM nominative)
FEM feminine)gender	GEN genitive) case
NEU neuter)	DAT dative)
STR strong declension	ACC accusative)
ATT attribute apposition	ADJ word class(adject./noun)
NUM number (singular/plural)	

(3) Even for restricted natural language, grammars are known to be extremely complex because of the multitude of syntactic aspects to be observed. The application of features simplifies the grammar insofar as it can be arranged in two levels,

a) a context-free grammar in terms of the variables from (1);

b) a feature program to be associated with each production on level a).

Example: Typical productions of level a) are

ME -> ME ME	ME -> QU ME
ME -> RE	SA -> ME sind ME?
ME -> RE ME	SA -> Sind ME ME?
ME -> RE IN	

The production

$ME_1 \rightarrow ME_2\ ME_3$

refers to the following feature program (syntactic variables are numbered for reference).

Part 1: Test of right-hand features for acceptance (reduction takes place only if the condition is true).

test (ME_2,+ADJ+ATT) ∧ test (ME_3, -ADJ-ATT)
∧ meq (MAS,FEM,NEU,ME_2,ME_3) ∧ meq (NOM,GEN,DAT,ACC,ME_2,ME_3)
∧ equ (NUM,ME_2,ME_3)

Part 2: Assignment of features to the syntactic variable on the left-hand side.

-ADJ-ATT, cop (NUM,ME_2),
and (MAS,FEM,NEU,ME_2,ME_3), and (NOM,GEN,DAT,ACC,ME_2,ME_3)

Feature operators are underlined. For example, test is true when the features of the first argument meet the condition specified by the second argument. meq is true whenever at least one of the listed features agree in both syntactic variables specified. cop copies the features of the syntactic variable specified.

3.4 Pharmacology

The natural language level is supposed to serve a variety of application areas. We postulate that these application areas are all served by the same natural language grammar since each must be explainable in terms of set theory. Consequently, these areas differ only in the vocabulary they assign to the object types. Level 3 is reached from level 2 simply by introducing names, and relating them to the object types. Below a few typical examples of assignment are given in the area of pharmacology.

proper names	medications, substances, companies, ailments, e.g. Thebacon, Morphium, CIBA, Angina pectoris
attributes	properties e.g. Tablette, rezeptpflichtig
references	e.g. Indikation and Kontraindikation (from ailment to medication), Hersteller (from company to medication)
references to measures	e.g. Preis, Dosis, Haltbarkeit
numbers or measures	e.g. 5 DM, 2 Tabletten/Tag, 4 Wochen
sentences	e.g. Welche Preise haben Praeparate, die bei Angina Pectoris indiziert sind und deren Kontraindikation nicht Glaukom ist?

3.5 Translations

The path between adjacent nodes is traversed by translation (sec.2.3, (3) and (4)). We shall briefly illustrate this for the passage between natural and set language. In this case translation consists of the three traditional phases: lexical analysis, syntactic analysis and code generation. The sentence
"Welche Firmen sind Hersteller tablettenfoermiger Medikamente?"
shall serve as an example.

Lexical analysis

Lexical analysis includes the mapping from the pharmacological to the natural language level, and for each word encountered, with a few exceptions, proceeds in three steps:

(i) reduction of a word to its word stem;

(ii) dictionary lookup resulting in a syntactical variable, values of some of its features, and a morphemic class, as well as the set level name for the word.

(iii) assignment of further features on the basis of the morphemic class and the actual morphemic ending.

The lexical analysis of the entire sentence results in

word	syn. var	features	int.name
Welche	QU	+MAS+FEM+NEU -NUM+NOM+ACC	DB
Firmen	ME	FEM-NUM+NOM+GEN+DAT+ACC	M26
sind	-	-	-
Hersteller	RE	+MAS+NUM+NOM+DAT+ACC	R23
	RE	+MAS-NUM+NOM+GEN+ACC	
tabletten-	ME	+MAS+NUM+NOM+ATT+STR+ADJ	
foermiger	ME	+FEM+NUM+GEN+DAT+ATT+STR+ADJ	M9
	ME	+MAS+FEM+NEU-NUM+GEN+ATT+STR+ADJ	
Medika-mente	ME	+NEU-NUM+NOM+GEN+ACC	M22
?	-	-	-

Note the syntactic ambiguities due to the different feature combinations for 'Hersteller' and 'tablettenfoermiger'. Note also that lexical analysis by itself cannot always determine the case (as for 'Firmen', all four cases are still possible), or the gender (as for 'tablettenfoermiger').

Syntactic analysis

Syntactic analysis includes three phases: reduction (level a)), feature analysis (level b)), final code manipulation. For each production applied, reduction and feature analysis follow each other immediately. Hence a production is applied in three steps:

(i) Matching of input string and right-hand side.
(ii) Test of right-hand features for acceptance.
(iii) If true, reduction to left-hand side and assignment of features.

For example, the production and feature program from sec.3.3 result in the following when applied to the phrase "tablettenfoermiger Medikamente":

ME2 (´tablettenfoermig´):
1) +MAS+NUM+NOM+ATT+ADJ (rejected on meq)
2) +FEM+NUM+GEN+DAT+ATT+ADJ (rejected on meq)
3) +MAS+FEM+NEU-NUM+GEN+ATT+ADJ
ME3 (´Medikamente´)
1) +NEU-NUM+NOM+GEN+ACC
ME1 (result):
1) +NEU+GEN-NUM-ADJ-ATT
(note the disambiguation)

The syntactic analysis of the entire sentence is illustrated in figure 3. Because of the possibility of ambiguities the result is a parsing graph rather than a tree (in this case the ambiguity of the sentence is due to ´Hersteller´). The numbers adjacent to the syntactic variables refer to an associated list of features.
Final code manipulation is left to the final stages of code generation, but must be considered part of the syntactic analysis because without it context-sensitive or transformational rules could not be avoided.

Code generation

Whenever a production is applied, a semantic action associated with it generates a functional set expression. Its arguments point to other such expressions unless they are individuals.
Example:

(tablettenfoermiger Medikamente)
M ∩ (,)
Mw(M9) Mw(M22)
(tablettenfoermig) (Medikament)

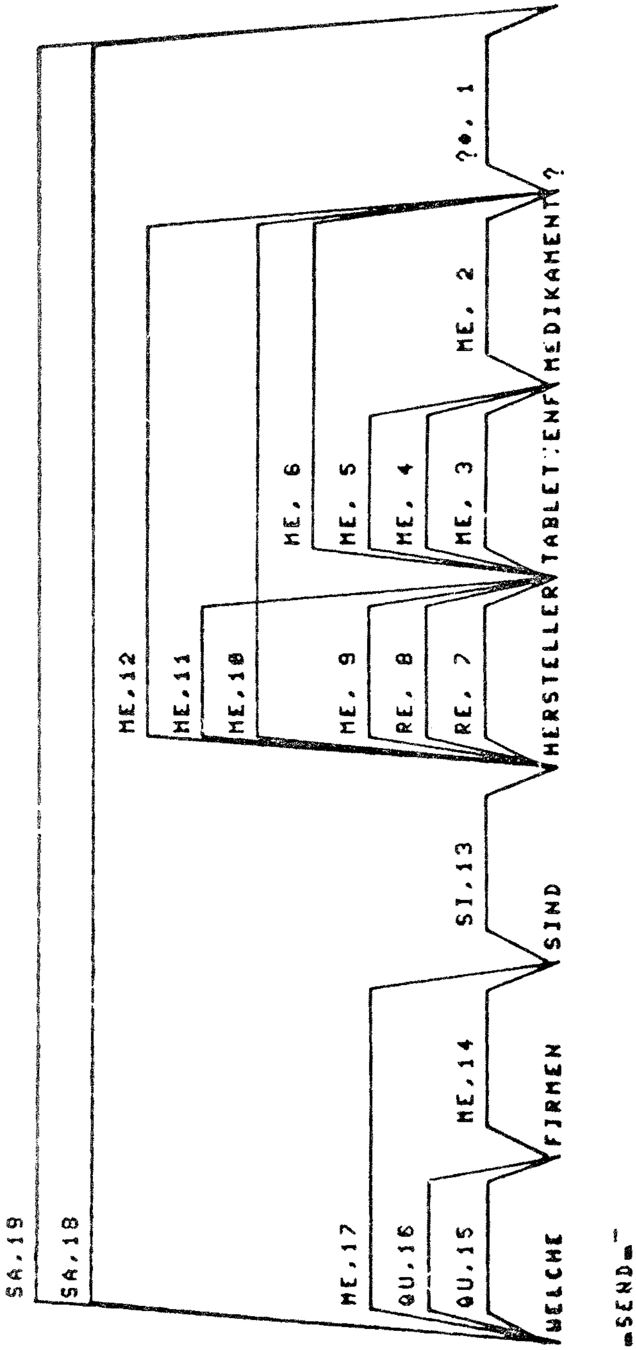

SA.19
SA.18
ME.17
QU.16
QU.15
WELCHE
ME.14
FIRMEN
SI.13
SIND
ME.12
ME.11
ME.10
ME. 9
RE. 8
RE. 7
HERSTELLER
ME. 6
ME. 5
ME. 4
ME. 3
TABLETTEN
ME. 2
MEDIKAMENT
?
?●. 1
"SEND"

Figure 3

WELCHE FIRMEN SIND HERSTELLER TABLETTENFOERMIGER MEDIKAMENTE ?

02300047	15000000	DB	(
10000001	01100025	X1	AA
15000000	14000005	(	M-T (5)
01100033	15000000	MW	(
04000032	16000000	M26	)
16000000	01200001	)	E
15000000	10000001	(	X1
01100025	15000000	AA	(
14100026	01100045	M-T (22)	MV*
15000000	01200040	(	VG*
15000000	01100030	(	EN
15000000	05000027	(	R23
16000000	01200044	)	MD
15000000	01100033	(	MW
15000000	04000033	(	M27
16000000	01100033	)	MW
15000000	04000026	(	M22
16000000	16000000	)	)
16000000	16000000	)	)
16000000	16000000	)	)
26000000	00000000	EWIRBE	--------

Figure 4

On completion of the parse, the pointer structure corresponding to the syntactic variable SA is transformed into a linear string. This string must be submitted to a further string manipulation for two reasons.

(1) Completion of the syntactic analysis.
Quantifiers do not yet appear in front of the expression. Moving them there is subject to a number of rules that govern their sequence.

(2) Optimization.
In many cases quantifiers (whose evaluation may be time-consuming) can be replaced by standard set or relation operators, e.g. DB by M∩.

The code resulting from translation of the sentence above is shown in the printout in figure 4.

Reverse translation

Set level names may immediately be translated into the pharmaceutical level simply by again invoking the dictionary. However, under certain conditions (empty sets) set expressions may themselves be part of a result. This requires a translation into both level 2 and level 3.
Examples:

Vg(R12, I14) -> Heilmittel fuer Psychosen
Mw(M9) -> tablettenfoermig
I2 -> Verophen

4 Semantic primitives as a basis

4.1 Motivation

In order to study the adequacy of the rules of ch.2 and to determine whether they must be further refined or augmented it is helpful, short of constructing systems, to examine existing systems that are arranged in the form of layers. One of the oldest systems of this kind (though it was not conceived that way) is Woods' question-answering machine [Wo 68, Wo 73]. Like the set theoretic approach, Woods' universe is composed of objects and interrelationships between them. Unlike the previous approach, these are not collected into mathematical sets and relations but treated as propositions to which a procedural approach is taken. This is probably due to an orientation towards explaining the semantics of natural language rather than manipulating concrete data bases.

4.2 Semantic primitives

Object_types

O   Elementary objects, e.g. Boston, AA-57, DC-9, 8:00 a.m.

F^n n-ary functions (n>1), e.g. departure time (of flight x_1 for place x_2). These need not be functions in the strict sense. If a function may yield more than one value (e.g. officer of a ship) it is defined as a successor function such that

(start)	officer(x,O) = a_1
	officer(x,a_1) = a_2
	. . . .
(end)	officer(x,a_n) = END

R^n n-ary relation (predicate) ($n \geq 1$), e.g. jet (flight x_1 is a jet), arrive (flight x_1 goes to place x_2).

D   Designators are either names of elementary objects or of the form $F^n(x_1,\ldots,x_n)$ where x_i is a designator; e.g. departure time (AA-57, Boston) for 8:00 a.m.

P   Propositions $R^n(x_1,\ldots,x_n)$ where x_i is a designator; e.g. jet (AA-57), place (Boston), arrive (AA-57, Chicago).

B   Truth values

Example: A set of semantic primitives for the flight schedules table (from [Wo 68]):

Primitive Predicates

CONNECT (X1, X2, X3)	Flight X1 goes from place X2 to place X3
DEPART (X1, X2)	Flight X1 leaves place X2
ARRIVE (X1, X2)	Flight X1 goes to place X2
DAY (X1, X2, X3)	Flight X1 leaves place X2 on day X3
IN (X1, X2)	Airport X1 is in city X2
SERVCLASS (X1, X2)	Flight X1 has service of class X2
MEALSERV (X1,X2)	Flight X1 has type X2 meal service
JET (X1)	Flight X1 is a jet
DAY (X1)	X1 is a day of the week (e.g.Monday)
TIME (X1)	X1 is a time (e.g. 4:00 p.m.)
FLIGHT (X1)	X1 is a flight (e.g. AA-57)
AIRLINE (X1)	X1 is an airline (e.g.American)
AIRPORT (X1)	X1 is an airport (e.g. JFK)

CITY (X1)	X1 is a city (e.g. Boston)	
PLACE (X1)	X1 is an airport or a city	
PLANE (X1)	X1 is a type of plane (e.g. DC-3)	
CLASS (X1)	X1 is a class of service (e.g. first-class)	
AND (S1, S2)	S1 and S2	(where S1 and S2 are propositions)
OR (S1, S2)	S1 or S2	
NOT (S1)	S1 is false	
IFTHEN (S1, S2)	if S1 then S2	

Primitive Functions

DTIME (X1, X2)	the departure time of flight X1 from place X2
ATIME (X1, X2)	the arrival time of flight X1 in place X2
NUMSTOPS (X1,X2,X3)	the number of stops of flight X1 between place X2 and place X3
OWNER (X1)	the airline which operates flight X1
EQUIP (X1)	the type of plane of flight X1
FARE (X1,X2,X3,X4)	the cost of an X3 type ticket from place X1 to place X2 with service of class X4 (e.g. the cost of a one-way ticket from Boston to Chicago with first-class service)

Operators

To every function and relation there exists a programmed subroutine (procedure) which determines a value of a function or the truth of a proposition.

Examples (procedure names are capitalized):

JET (AA-57) -> true

ARRIVE (AA-57,Chicago) -> true

ARRIVE (AA-57, Boston) -> false

DTIME (AA-57, Boston) -> 8:00 a.m.

Whereas the abstract machine of ch.3 was based on object types but specific operators, the abstract machine in this case is defined in terms of both object and operator types. Specific instances must be supplied by the user for both of them. However, with the advent of microprogramming, computer scientists should have little problems in adjusting to this kind of notion.

Control mechanism

As in the preceding example, programs are expressed in functional notation, e.g.

TEST(CONNECT (AA-57, BOSTON, CHICAGO))
would stand for "Does AA-57 go from Boston to Chicago?". Likewise, queries of any appreciable degree of complexity are based on the notion of bounded quantifier as a representative for loops.

The format for a quantified expression is

FOR <quant> <var>/<class>:<pvar>; <qvar>
where

<quant>	a type of quantifier (EACH,EVERY,SOME,THE, nMANY).
<var>	a bound variable.
<class>	class of objects over which quantification is to range. The specification is performed by special enumeration functions, e.g. SEQ,DATALINE,NUMBER,AVERAGE. Besides enumeration these functions may perform searches or computations.
<pvar>	restriction on the range } may both be quantified expressions.
<qvar>	scope }

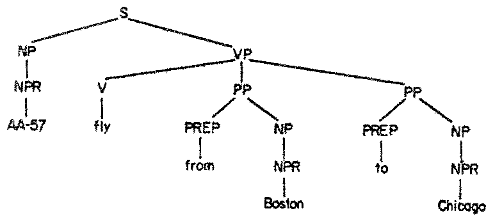
Unlike KAIFAS where the result of the evaluation of an expression is automatically retranslated and displayed, this must be explicitly requested by commands such as TEST (test truth of a proposition), PRINTOUT (print the representation for a designator).

Examples:
(FOR EVERY X1 / (SEQ TYPECS):T; (PRINTOUT (X1))
prints the sample numbers for all the lunar samples which are of type C rocks, i.e. breccias (T stands for "true").
(TEST (FOR 30 MANY X1 / (SEQ FLIGHT):JET(X1); DEPART (X1,BOSTON)))
"Do 30 jet flights leave Boston?"

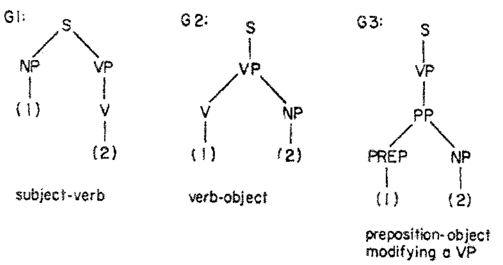
4.3 Natural language

As a general rule, the introductory remarks to sec.3.3 apply here as well: The level of the "English-like" query language provided on level 2 is influenced by the range of expressions possible on the previously discussed level 1. In contrast to KAIFAS, inspection of the data base is not limited to the evaluation of level 1 expressions but may take place during translation from level 2 into level 1, too. The semantic actions associated with a rule of grammar impose further restrictions, e.g. they make sure that the first argument of CONNECT is indeed an instance of the class FLIGHT.

This is illustrated by the following example. In a first step a syntactic analysis is performed and a phrase marker is derived, e.g.



Since verbs in English correspond roughly to predicates, and noun phrases are used to denote the arguments of the predicate, the verb in the phrase marker will be the primary factor in determining the predicate. In the example, the predicate will be CONNECT. For this it is necessary that the subject be a flight and that there be prepositional phrases whose objets are places representing origin (from) and destination (to). The grammatical relations among elements of a phrase marker are defined by partial tree structures, e.g.



Among the three structures, G1 and G3 both match subtrees in the phrase marker. Which of these is acceptable depends on the additional rules, e.g.

(G1:FLIGHT(1) and(2) = fly).

((1) and (2) are positional variables in the partial tree structure). This rule obviously is satisfied. More complex rules are possible; for example, the topmost S-node of the phrase marker is matched by the rule

1-(G1:FLIGHT((1)) and (2) = fly) and
2-(G3:(1) = from and PLACE ((2))) and
3-(G3:(1) = to and PLACE((2)))
==> CONNECT(1-1,2-2,3-2)

4.4 Airline guide

The system under discussion was first applied to a flight schedules table. To illustrate the application interface, a few examples of queries shall be given below (from [Wo 68]).

Does American Airlines have a flight which goes from Boston to Chicago?
What is the departure time from Boston of every American Airlines flight that goes from Boston to Chicago?
What American Airlines flights arrive in Chicago from Boston before 1:00 p.m.?
How many airlines have more than 3 flights that go from Boston to Chicago?

4.5 Lunar geology

More recently the system has been applied to access, compare and evaluate the chemical analysis data on lunar rock and soil composition that was accumulating as a result of the Apollo missions [Wo 73]. Examples:

What is the average concentration of aluminum in high alkali rocks?
Give me all analyses of S10046!
How many breccias contain olivine?
Do any samples have greater than 13 percent aluminum?
What is the average model concentration of ilmenite in type A rocks?

4.6 Critique

(1) The possibility of inspecting the data base both on level 1 and during translation from level 2 to level 1 introduces a note of confusion. Since, according to sec.2.3, translation is directly related to definition, the translation process must make no reference to the data base. The lack of separation will have practical repercussions: Either certain changes on level 1 will necessitate changes in the rules of grammar, or parts of the control mechanism for level 1 must be duplicated for translation purposes.

(2) In Woods' system the subroutines do not appear to verify that their arguments are of the proper kind (e.g. ARRIVE does not check whether AA-57 is indeed a flight or Chicago a place), since this

is done on translation. If one left this (correctly) to level 1 then primitive predicates and functions are related to each other. These interdependencies may be expressed by a set of axioms, or in the parlance of data structures by types or categories corresponding to those unary predicates that restrict ranges of arguments. As a consequence, the concepts of abstract machine and relationships between abstract machines must account not only for primitive terms but for axioms as well. (Note that the KAIFAS machine circumvents this problem only by prescribing all operators.)

(3) Operators (subroutines) and objects are interdependent as well, albeit in a one-to-one fashion. In order to make sure that the requirements governing the relationship between abstract machines are met it suffices to treat a predicate or function and its corresponding procedure as two instances of the same resource.

5 Relational model

5.1 Motivation

One of the most widely discussed approaches to data bases is Codd's relational model [Co 70,Co 72, We 74] which lends itself particularly well to an interpretation by abstract machines. Codd supposes his users to explain their universe in terms of table-like structures. Intuitively speaking, a table consists of a number of entries that are formatted in exactly the same way: a sequence of fields ordered on certain headings or field names or, as they are called here, attributes. More formally, a entry is an ordered n-tuple and, consequently, a table is a relation that may be named. Entries are not named but are uniquely identified by a key, i.e. the contents of particular fields.

A certain familiarity with the relational model is assumed on the reader's part. Only its interpretation by a machine will be examined here.

5.2 Relational algebra

Objects

A attributes naming a set of objects (domain)

R^n relations

$R^n(A_1,A_2,\ldots,A_n) \subseteq A_1 \times A_2 \times \ldots \times A_n$
Example: SUPPLIER (SUPPLIERNR, NAME, LOC), KEY=SUPPLIERNR

SUPPLIER:	SUPPLIERNR	NAME	LOC
	1	Jones	New York
	2	Smith	Chicago
	3	Connors	Boston
	4	Thompson	New York

Key attributes are indicated; keys may be composite. Hierarchical and other relationships are usually eliminated by normalization. Hence all relations can be assumed to be normalized.

$T^n \in R^n$ n-tuple.

Operators [We 74]

Standard relation operators

$R^{n_1} \otimes R^{n_2} \rightarrow R^{n_1+n_2}$ Direct Product:
$\{(T^{n_1} \frown T^{n_2}) \mid T^{n_1} \in R^{n_1} \wedge T^{n_2} \in R^{n_2}\}$
($\frown$ Concatenation operator)

$R^n \cup R^n \rightarrow R^n$ Union
$R^n \cap R^n \rightarrow R^n$ Intersection
$R^n - R^n \rightarrow R^n$ Difference
(Union, Intersection, Difference: attributes must be "compatible")

Special operators

$R^n[A] \rightarrow R^m$ Projection: Relation R^n restricted to the attributes $A=\{A_1,\ldots,A_m\}$.

$R^{n_1}[A \theta B] R^{n_2} \rightarrow R^{n_1+n_2}$ Join:
$\{(T^{n_1} \frown T^{n_2}) \mid T^{n_1} \in R^{n_1} \wedge T^{n_2} \in R^{n_2} \wedge T^{n_1}[A] \theta T^{n_2}[B]\}$
where A,B sets of attributes, θ one of $\{=,\neq,<,\leq,>,\geq\}$.
(Slight modifications, e.g. natural join, are possible).

$R^n[A \theta B] \rightarrow R^n$ Restriction: $\{T^n \mid T^n \in R^n \wedge T^n[A] \theta T^n[B]\}$
where A,B,θ as above.

$R^n[A \div B] R^n \rightarrow R^m$ Division: [Co 71], p.74.

Control mechanism (Relational algebra)

Since all operators have been defined as infix operators, "programs" are formed by linear sequences of operators and operands rather than by nested expressions. For an example see sec. 5.3.

5.3 Relational calculus (ALPHA)

In place of relation algebra Codd proposes an applied predicate calculus (relational calculus), and proceeds to show that any expression in the relational calculus (alpha-expression) may be reduced to an equivalent relation algebraic expression.

Alphabet for the calculus:

Individual constants, $a_1, a_2, a_3, \ldots$

Index constants, 1, 2, 3, 4, ...

(attributes are indexed per relation instead of named)

Tuple variables, $r_1, r_2, r_3, \ldots\ldots$

Predicate constants, monadic, $P_1, P_2, P_3, \ldots$;

dyadic, $=, \neq, <, \leqslant, >, \geqslant$

Logical symbols, $\exists, \forall, \wedge, \vee, \neg$

Delimiters.

Simple alpha-expressions have the form

$(t_1, t_2, \ldots, t_k) : w$

where - w a well-formed formula,

- t_i distinct terms consisting of an indexed or non-indexed tuple variable,
- the set of tuple variables occurring in $t_1, \ldots, t_k$ is precisely the set of free variables in w.

Example: Alpha-expression for "Find the name and location of all suppliers each of whom supplies all projects":

$(r_1[2], r_2[3])$:

$P_1 r_1 \wedge \forall P_2 r_2 \exists P_3 r_3 ((r_1[1]=r_3[1]) \wedge (r_3[3]=r_2[1]))$

After reduction to relation algebra:

$S_1 = R_1$

$S_2 = R_2$

$S_3 = R_3$

$S = S_1 \otimes S_2 \otimes S_3$

$T_3 = S[1=6] \cap S[8=4]$

$T_2 = T_3\ [1,2,3,4,5]$

$T_1 = T_2\ [(4,5) \div (1,2)] S_2$

T = $T_1[2,3]$

ALPHA is a language for alpha expressions that is slightly more appealing to the user than the predicate form shown above. The example may be reformulated in ALPHA as

```
RANGE SUPPLIER L
RANGE PROJECT P
RANGE SUPPLY K
GET W  (L.NAME, L.LOC):
          (∀P)(∃K) ((L.SUPPLIERNR=K.SUPPLIERNR) ∧ (K.PROJNR=P.PROJNR))
```

or, equivalently (order of quantifiers must be maintained!),

```
RANGE SUPPLIER L
RANGE PROJECT P ALL
RANGE SUPPLY K SOME
GET W (L.NAME, L.LOC):
         (L.SUPPLIERNR = K.SUPPLIERNR) ∧ (K.PROJNR = P.PROJNR)
```

5.4 Higher levels

For reasons similar to the ones in chs. 3 and 4 languages have been devised that do not have to rely on a user's formal training. One language of this kind is SQUARE [Bo 74] which has been shown to be reducible to the relational calculus. However, the view of relations offered by SQUARE is different from that offered by ALPHA:

(i) Scan a column or columns of a table looking for a value or a set of values (as opposed to inspecting one row after another).
(ii) For each such value found examine the corresponding row and elements of given columns in this row.

SQUARE statements are of a form such as ("disjunctive mapping")

$_{B}R_{A}(S)$

(read: "find B of R where A is S") that defines a mapping such that R is a relation, A and B are sets of attributes (domain and range, respectively), S is an argument that may itself be an expression. Other forms, e.g. for projection, conjunctive and n-ary mappings, have a similar appearance.

Example: $_{NAME}EMP_{DEPT}$('TOY')
stands for "Find the names of employees in the toy department".

More recently attempts have been reported that allow a user to engage a relational data base system in a dialog founded on natural English [Co 74]. The approach differs drastically from the ones discussed in chs.3 and 4 in that a truly two-way communication is envisioned.

5.5 Comment

It has been shown that both ALPHA and SQUARE are equivalent to the relational algebra, i.e. any query expressible in relation algebra is expressible in ALPHA and in SQUARE, and vice versa. Hence ALPHA and SQUARE are themselves equivalent. Equivalence is a symmetric relation. The condition on the succession of abstract machines does not preclude equivalence, the definition of the hierarchy by restriction however does. From the point of user sophistication a hierarchy could still be given as relational algebra - ALPHA - SQUARE (in the direction of increasing level). This indicates that further refinement on the notion of hierarchy is necessary.

6 Conclusions

There are some striking similarities between the examples of chs.3,4 and 5:

- In each the lowest level has been well formalized.
- All rely on quantification as a means for building complex expressions.
- All tend towards natural language on their higher levels.
- All three systems have been implemented and found some application.

On the other hand, only one of them (ch.5) so far attempted to provide a less formal but still stylized language on an intermediate level. Experiences indicate that, at least in some well-defined situations, this may be necessary with the KAIFAS system (ch.3) as well.

While a few examples do not constitute proof, at the very least they do suggest that hierarchies of user languages could meet the objectives mentioned in the introduction. Of course, the relationship between successive levels will have to be made much more precise, as has been indicated before. Furthermore, higher levels imply a number of successive translations, and techniques must be explored to measure and perhaps raise the efficiency of higher levels. Finally, the paper did not attend to the critical question what form the root should take; this appears to be a largely unsolved problem.

Acknowledgement. The author is grateful to G.Goos for carefully reading the manuscript and making helpful suggestions.

References

[Ab 74] J.R.Abrial, Data Semantics, in [Kl 74], 1-59

[Bo 74] R.F.Boyce, D.D.Chamberlin, W.F.King, M.M.Hammer, Specifying Queries as Relational Expressions, in [Kl 74], 169-176

[Bu 72] Burroughs Corp., B6700/7700 Executive System Programming Language (ESPOL), Information Manual, 1972

[Co 70] E.F.Codd, A Relational Model for Large Shared Data Banks, Comm.ACM 13(1970), No.6, 377-387

[Co 72] E.F.Codd, Relational Completeness of Data Base Sublanguages, in: R.Rustin (ed), Data Base Systems, Courant Computer Science Symp., Prentice-Hall, Inc. 1972, 65-98

[Co 74] E.F.Codd, Seven Steps to Rendezvous with the Casual User, in [Kl 74], 179-199

[Col 68] L.S.Coles, An Online Question-Answering System with Natural Language and Pictorial Input, Proc. 23rd Natl. ACM Conf. (1968), 169-181

[Go 73] G.Goos, Hierarchies, in F.L.Bauer (ed), Advanced Course on Software Engineering, Lecture Notes in Econ. and Math. Systems, vol.81, 29-46

[Gr 69] C.C.Green, The Application of Theorem Proving to Question-Answering Systems, Tech. Rep. No. CS138, Stanford Univ. 1969

[Kl 74] J.W.Klimbie, K.L.Koffeman (eds), Data Base Management, North-Holland Publ. Co. 1974

[Kr 75] K.D.Kraegeloh, P.C.Lockemann, Hierarchies of Data Base Languages: An Example, Information Systems (in print)

[Su 74] B.Sundgren, Conceptual Foundation of the Infological Approach to Data Bases, in [Kl 74], 61-94

[SI 74] ACM SIGPLAN Symposium on Very High Level Languages, March 1974, ACM, New York 1974

[We 74] H.Wedekind, Data Base Systems I, BI-Wissenschaftsverlag, Reihe Informatik, vol.16, 1974 (in German)

[Wi 68] N.Wirth, PL360, A Programming Language for the 360 Computers, Journ.ACM 15(1968), No.1, 37-74

[Wo 68] W.A.Woods, Procedural Semantics for a Question-Answering Machine, Proc. AFIPS Fall Joint Comp.Conf. 33(1968), 457-471

[Wo 73] W.A.Woods, Progress in Natural Language Understanding - An Application to Lunar Geology, Proc. AFIPS Natl.Comp.Conf. 42(1973), 441-450