ERNEST: A Semantic Network System for Pattern Understanding

HEINRICH NIEMANN, MEMBER, IEEE, GERHARD F. SAGERER, MEMBER, IEEE, STEFAN SCHRÖDER, AND FRANZ KUMMERT

Abstract—**This paper gives a detailed account of a system environment for the treatment of general problems of image and speech understanding. It provides a framework for the representation of declarative and procedural knowledge based on a suitable definition of a semantic network. The syntax and semantics of the network are clearly defined. In addition, the pragmatics of the network in its use for pattern understanding is defined by several rules which are problem independent. This allows one to formulate problem-independent control algorithms. Complete software environments are available to handle the described structures. The general applicability of the network system is demonstrated by short descriptions of three applications from different task domains.**

Index Terms—**Graph search, image understanding, knowledge acquisition, problem-independent control, semantic network, speech understanding, system shell.**

I. Introduction

THERE has been an increasing interest in the automatic interpretation of sensor signals like images, image sequences, or continuous speech. The goal is to compute a symbolic description of those aspects or contents of the signal which are relevant in a particular application; this may be viewed as a general problem of pattern recognition. It is generally agreed that this requires the acquisition, representation, and use of task-specific knowledge by the system. Processing proceeds from the sensor signal, represented by an array of integer sample values, via different levels of more and more abstract representation of the content of the signal. Representational levels in image understanding may be, for example, edges and regions obtained from an initial segmentation, three-dimensional surfaces, symbolic names of objects and their relations, conceptions of motion like a "heart cycle" or a "pedestrian crossing a street," interpretations concerning a diagnostic description like "hypokinetic motional behavior," concerning a situation like "congested highway," or concerning an event like "placing a part into a device for assembly." These examples demonstrate that image understanding requires representation of knowledge about quite different aspects of a sensor signal and on different levels of abstraction. Several systems have been designed and realized demonstrating the feasibility of knowledge-based processing of image or speech signals; for example, see the systems described in [4], [11], [27], [31], [33], and [41].

A system capable of performing the above-mentioned processing basically consists of the following four components [29].

1) Methods for low-level preprocessing and initial segmentation. This has usually been done so far without the employment of explicitly represented knowledge, although there are some exceptions to this, for example, [28].

2) Knowledge for doing task-specific recognition and understanding of the segmentation results.

3) Control for determining a processing strategy by activating the appropriate algorithms at the proper time using a relevant subset of the available data.

4) Results database for storing the results of processing and making them available to processing algorithms as necessary.

It may be convenient to also have the following three additional components in a system.

5) Knowledge acquisition for automatic learning of the task-specific knowledge.

6) Explanation for a convenient assessment and visualization of system activities and resources.

7) User interface to make system resources and capabilities transparent also for the user who does not know the details of the system.

There are several textbooks covering various topics involved in the above system structure, for example, [1], [10], [11], [24], [29], [38], [45].

In this paper, we will outline a particular system shell, called ERNEST, for knowledge representation which is based on the ideas developed in [2], [20], and [48] concerning semantic networks. The general background is discussed in Section II. A first realization is described shortly in [31] and in more detail in [41]. From this resulted an extended and improved definition of the structure of and the inferences for the semantic network which is described in Section III. It allows the integration of a complete system for image or speech understanding in a homogeneous network structure [32], [34]. An essential prerequisite for such a homogeneous system is the possibility to derive task-independent control algorithms; two realized algorithms are outlined in Section IV. The ideas

Manuscript received January 4, 1988; revised January 8, 1990. Recommended for acceptance by R. De Mori.

The authors are with the Lehrstuhl für Informatik 5 (Mustererkennung), Friedrich-Alexander-Universität Erlangen-Nürnberg, D-8520 Erlangen, West Germany.

IEEE Log Number 9036248.

presented in Sections III and IV were realized in a versatile software system to give the user a convenient working environment as outlined in Section V. This system shell ERNEST also includes tools for knowledge acquisition and explanation. But these two system components will not be discussed in this paper. In order to restrict the topics, we concentrate on knowledge representation and utilization. Finally, in Section VI, we show with three examples from different task domains that the ideas are powerful and feasible, and we close the paper by an outlook on further work in Section VII.

II. Background

The problem of knowledge representation in general is deeply discussed in a number of papers and textbooks, e.g., [10], [47], [22], [23]. Within the topic of this paper, a short discussion on special considerations related to pattern understanding will be presented. First of all, different types and components of knowledge for pattern understanding will be separated. In order to make a decision for a basic approach for the desired system shell, a few words on adequacy criteria for knowledge representation languages are necessary. These criteria, together with the remarks on different schemes and languages, form the decision to choose the semantic network approach for the pattern understanding shell ERNEST. The very basis of all the following ideas and discussions is the general system architecture consisting of those components which were enumerated in the Introduction.

A. Knowledge Components and Types

In order to achieve automatic descriptions of patterns, different kinds of knowledge must be considered. First of all, there are objects, events, and other problem-specific knowledge concepts which must be modeled. This *type* of knowledge is often called the declarative part of a knowledge base. Such models can be used for the interpretation of patterns. Therefore, it is necessary to know how they can be used. The knowledge about the use of a declarative model builds up the inference processes. This is a second type, which we call procedural knowledge. Additionally, structures describing signal areas are created during the analysis of patterns. These are summarized by the term *a posteriori* data and are built of instances. They connect signal areas with concepts of the declarative knowledge. So far, all three types of related to symbolic descriptions of patterns. But this kind of description is only one *component* of knowledge in a pattern analysis system. Because we are working with patterns, i.e., with sensor signals, other components must be taken into account. Signals must be transferred into symbolic descriptions. Each declarative concept is associated with numerical or symbolic features or attributes. These features must be defined for the concepts, and we need for each such feature special algorithms to detect it, and functions to combine features to new ones. Therefore, a knowledge base must contain an attribute component. Like the symbolic component, it is divided into three different types of knowledge and data. The declarative type defines the attributes for the concepts by their domain and their content. The procedural type addresses the functions which are needed to calculate and to combine the attributes. The data type shows values of attributes in the context of instances. A third component is based on the fact that signals are noisy. They are a source of errors and ambiguity. No decision in terms of "yes" or "no" is possible, but every decision is judged with some kind of certainty or possibility. In order to allow the combination and the comparison of such judgments of different intermediate results, a calculus for judgments must be fixed for a pattern analysis system. This fact should not imply that all applications realized with the same system shell must use the same judgment calculus. The shell must allow the definition of a calculus for an application. This definition is similar to the attribute and the symbolic description component. If a calculus is chosen, the declarative type is given by the data structures which are able to store the judgments. Furthermore, the calculus gives the procedural knowledge type by the functions to combine and to compare judgments. The *a posteriori* type are the values of judgments for symbolic descriptions and for attributes. The ideas about the three types and the three components of knowledge as described above are summarized in Fig. 1. Each of the components consists of the types declarative knowledge, procedural knowledge, and results. The system component "knowledge" of a pattern understanding system must cover both procedural and declarative knowledge for all three knowledge components concepts, attributes, and judgments. The "result database" stores the *a posteriori* data of all these knowledge components. The system component "control" only uses the two knowledge types and the results to guide the analysis processes.

So far, it is shown what components and types of knowledge a pattern analysis system has to take into account. It depends on the organization of the knowledge, and how complex the retrieval and the activation of the different types and components are for the component "control." This yields the problem of dependencies and relationships between components and types. There are several approaches which will be discussed in more detail in Section II-B. A first decision for the organization of a knowledge base is whether the procedural or the declarative knowledge should be the main items. Or in other words, should procedures dominate the declarative knowledge or vice versa? A second decision addresses the components. For example, in classical pattern recognition systems, the judgments, i.e., the classifier, dominate the attributes, and the concepts are only used in an implicit manner. Contrarily, in most pattern analysis systems, the judgments are subordinated to the attributes and concepts. Also, the relationships between attributes and concepts can be realized in two different ways. Features can be associated with special functions having concepts as arguments or properties of concepts. In the first case, concepts depend on attributes. They can be a member of the do-

declarative knowledge	procedural knowledge	results
concepts	inferences	instances
features	extraction,combination	feature values
scheme for judgements	calculation and combination scheme and functions	values

Fig. 1. Three types of knowledge (horizontal direction) and three components of knowledge (vertical direction) for pattern understanding.

main of the function which represents the attribute or they are not. In the second one, attributes are defined in the context of one or more concepts. They are attributed to the concepts and have a role in the definition of a concept. The resulting questions are as follows.

Should knowledge representation for pattern analysis use the procedural or the declarative point of view? What component is used for the main entities of the knowledge base, the symbolic one, the attributes, or the judgments? Which structures and procedures must the "control" component know to fulfil his task?

In order to give help in answering these questions, a number of adequacy criteria for knowledge representation schemes and languages were introduced in the literature. Although these criteria do not allow us to measure the adequacy of a knowledge representation approach in a quantitative way, they give indications for examining or at least discussing the quality of the organization tools for knowledge bases.

B. Criteria for Calling a Knowledge Representation Scheme Adequate

In [22], [2], and [42], the *epistemological adequacy* is accentuated as the main criterion for knowledge representation languages. A compact definition for this criterion is given in [22]: "A representation is called epistemologically adequate for a person or a machine if it can be used practically to express the facts that one actually has about the aspects of the world." Brachman [2] pointed out that an epistemologically adequate scheme must be neutral with respect to a conceptional level of a knowledge base. This level is built of those concepts and the relationships between them, which are relevant for a given task domain. Therefore, a representation scheme should be independent of applications. Other criteria address the logical completeness and decidability, the algorithmic complexity, ergonomical comprehensibility, and psychological problems [42]. All of these criteria do not only ask questions concerning the declarative types of knowledge. Knowledge representation requires syntax, semantics, and pragmatics of an artificial language. The definition of pure syntactical structures—like a formal language or data structures—is not sufficient. The definition of both an interpretation and an interpreter is necessary. For all knowledge representation languages, especially if they are used for the interpretation of sensor signals, one should take a further criterion into account: the handling of uncertain data, and therefore uncertain decisions. In the present state of the art, it is impossible to extract unique symbolic attributes out of sensor signals, take them as being correct, and finally run symbol to symbol inferences as is done in "standard" expert systems. On the contrary, sensor signals are a source of error and uncertainty. Parameters or initial symbolic descriptions derived from the signal can only be scored or judged with respect to one calculus like probabilities or fuzzy logic. Therefore, besides uncertain knowledge of a task domain, uncertain input data also must be handled by the representation language. Intermediate results must be judged, and the language has to offer the possibilities to do this job. Because of these facts, we introduce the criterion *adequacy for handling uncertainty* for both knowledge and data. Examples for knowledge bases which handle both kinds of uncertainty are given, for example, in [5], [46], and [41].

C. Knowledge Representation Schemes and Languages

Following [26], knowledge representation schemes can be classified into the categories logical schemes, procedural schemes, and semantic networks. But most knowledge representation languages, e.g., PROLOG or PSN, subsume more than one of these categories. The classification problem is described in [26]: "When trying to classify representation schemes, we consider the world as a collection of individuals and as a collection of relationships that exist between them. The collection of all individuals and relationships at any time in any one world constitutes a state, and there can be state transformations that cause the creation/destruction of individuals or that can change the relationship among them."

One example to illustrate this is the language PROLOG [17]. From one point of view, PROLOG is a logical representation scheme. Therefore, it employs the notions of constant, variable, function, predicate, logical connective, and quantifier in order to represent elementary facts. A knowledge base is a collection of terms and formulas. The sequence

$$\text{``}B_1 \wedge B_2 \wedge \cdots \wedge B_m \rightarrow A\text{''}$$

of symbols represents a well-formed formula. No information about the interpretation of such a formula is given. It is just a sequence of symbols which satisfy rules for building up syntactically correct sequences. Such well-formed formulas are treated by a dual semantics. Besides the traditional Tarskian semantics

"B_1 and B_2 and $\cdots$ and B_m implies A"

the procedural semantics

"if you want to establish A, try to establish B_1 and B_2 and $\cdots$ and B_m"

is used. The Tarskian semantics gives an *interpretation* of the formula, while the procedural semantics offers an *interpreter*. It shows how a formula can be used within an analysis of a knowledge-based system. The static Tarskian interpretation is enlarged by a procedural component. Formulas are viewed as facts and programs at the same time. The procedural semantics establishes a rough method for proving theorems, and is therefore comparable to the Gentzen calculus or the resolution method [21]. For the definition of a procedural semantics, the critical point is the use of stored knowledge, i.e., the definition of problem-independent inferences rules to construct a knowledge representation language based on a knowledge representation scheme. Because of this fact, we prefer the notation that a pragmatics must be defined for a scheme in order to get a language. The term "pragmatics" coincides with linguistic theory, which defines a language by the steps syntax, semantics, and pragmatics.

Procedural schemes view a knowledge base as a collection of active processes and agents. More or less, any programming language can be looked at as a procedural knowledge representation scheme. Schemes like production systems [13] and PLANNER [14] offer activation mechanisms for processes. In both schemes, a knowledge base is built of pairs. Each pair consists of a pattern and one or more actions which manipulate the working memory. If the pattern of a pair can be successfully matched to the database, the corresponding theorem in PLANNER or the action of the rule in a production system is executed. The PLANNER control module uses the backtracking algorithm. This is also used in many applications based on production systems. Nevertheless, numerous other algorithms are used to control the search in such systems.

Even though semantic networks are of large diversity, there exists a most basic form. *A priori* and *a posteriori* knowledge is expressed by nodes and directed labeled edges, called links. The nodes model concepts or classes of concepts or are descriptions of individuals. Edges are used to express binary relations between the nodes. Together with their associated edges, nodes which stand for concepts or classes of concepts build up the model for an application. They represent the *a priori* knowledge. *A posteriori* data which are generated during an analysis process are represented by individual nodes and by edges between individual nodes or between nodes for concepts and individuals. Whereas identical concepts are identified in formulas implicitly by utilization of identical names, a concept is represented once in a semantic network. All relationships with which the concept is associated are centered in the node standing for the concept. The main problem of the scheme was that most of the early languages had little or no semantics for the types of nodes and links they used. The necessity for a semantics and an epistemological adequacy was pointed out in [48], [2]. In order to define a unique interpretation of the different types of nodes and links in a language, this set must be restricted. On the other hand, to get an epistemologically adequate language, this set must be sufficient to build up knowledge bases for all possible or at least a large number of applications. Most network languages offer different organizational axes for structuring a knowledge base [10]. We call an axis a hierarchy in the network if it defines a partial nonreflexive order on the set of concepts. The most used axes are the following.

Classification: A real world object is associated with its generic type(s). This axis forces a distinction between a concept, which is the intentional description or a prototype of a concept, and an instance, which is a member of the extensional set of a concept. Some languages like PSN [20] use classification recursively with included cycles.

Aggregation: This type connects a concept or an instance with other concepts or instances, respectively, which describes their components or parts.

Generalization: This type relates a concept to more general ones. Generalization, often called is_a, defines a hierarchy in the network due to a partial order. In most approaches, attributes associated with a general concept are inherited by the more special ones, unless they are explicitly modified.

So far, semantic networks are a helpful scheme for an efficient organization of declarative knowledge bases and the corresponding results databases. A given semantics, i.e., a unique interpretation, for the different types of nodes and links is one condition to use a semantic network in a knowledge-based system. In order to build a network with respect to defining a language which can be used as a kernel for a complete system, data structures must be defined to cover the nodes and links. At this point, semantic network approaches like KL-ONE [3] and PSN [20] are influenced by the notation of frames [25]. Such frames are complex data structures for representing stereotypical informations for a task domain. A frame has slots for the objects which play a role in the situation, as well as conditions between the slots. Furthermore, processes and facts are attached to the slots.

In KL-ONE and PSN, frame-like data structures are used to build up the nodes of the semantic network. Links are described within such a data structure by slots of different types, one for each link type. The interpretation of the slots and their items is defined with respect to the semantics of the semantic network. While KL-ONE only uses a procedural attachment associated with the slots, in PSN a pragmatics (procedural semantics) is also defined. The slots in a concept are divided into prerequisites and consequences with respect to the instantiation procedure of the concept.

Asserting control to semantic network languages results in a system shell if a pragmatics is defined for the language. Therefore, they also propose a system architecture. The resulting database is structured along organizational axes, and inferences are created according to the pragmatics definition of the network language.

Because of the following facts which are based on the criteria in Section II-B and the claimed components for a knowledge base in Fig. 1, we decided to use a semantic network scheme for ERNEST (**Er**langen Semantic **Ne**twork **S**ystem and **T**ools). The node-centered representation of concepts supports the compact definition of knowledge bases and working memories. It helps in evaluating results of an analysis process. Therefore, semantic networks are at least near being ergonomically adequate. Procedural attachment is useful for binding algorithms for attributes to the concept, and subsequently to the attributes for which they are used. PSN shows one way to integrate inference mechanisms into the language definition of a semantic network. Running systems like ALVEN [46] indicate a possible algorithmic adequacy and the adequacy for handling uncertainty. There are semantic network languages like KL-ONE which were developed in order to be epistemologically adequate. Nevertheless, Schefe [42] shows an example, given in Section III-B, which can hardly be represented adequately in KL-ONE. This indicates that the set of the epistemological primitives is not sufficient in KL-ONE. Even if there is no proof for the logical adequacy of a semantic network language, there are some facts which should be mentioned. If semantic networks are a syntactical variant of the first-order predicate calculus [38], they are logically adequate. If they have additional properties, at least the restrictions of a network to this calculus is logically adequate. One concept is represented by exactly one node in a semantic network. For a translation of such a network into logical formulas, the concept has to be denoted in as many formulas as the links which connect the concept with other ones. Because of this fact, McDermott [23] called a semantic network "predicate calculus plus index-scheme." If a large knowledge base is necessary for some applications, it will be easier to check a few hundred concepts compared to a few thousand rules, theorems, or predicates.

III. The Semantic Network in ERNEST

After the general discussion of knowledge representation in the previous section, we now turn to the presentation of the particular knowledge representation language developed in ERNEST. There is a clear distinction of the syntax, semantics, and pragmatics of the network (not of the task domain!), and these aspects are described in the three subsections to follow. By "syntax of the network," we mean the available data structures and the necessary restrictions without regard to their relation to a particular meaning of these structures. We think that it is useful to start with a presentation of the syntax because it gives a short overview of the relevant components of the network. The meaning of the data structures, in particular of the nodes, links, and substructures as well as their slots and items, as used here, is described in the "semantics of the network." It is important to note that a semantic network in an image or speech understanding system not only is to represent some declarative and procedural knowledge, but also has to provide the basis for its utilization for knowledge-based signal understanding. This aspect is described as the "pragmatics of the network" in the third subsection.

A. The Syntax of the Network

Having decided upon a certain definition of a semantic network approach, this definition may need extension, modification, crispening, and improvement as new insights emerge. Therefore, our definition of a syntax of the network proceeded mainly in two steps. The first step only will be considered very briefly. It consisted of an experimental environment allowing an easy definition and modification of different semantic network structures; details of this first step are given in [7]. This allowed a user to define and test different structures of a semantic net. From these tests and also from experiences obtained in different applications of the semantic network, a kind of default structure was derived in the second step. This default structure is intended for the user who wants to use a complete tool to solve a particular problem of image or speech understanding, but who does not want to experiment with different definitions of a semantic network. In the following, we will discuss only the structure derived in the second step.

Important problems in the definition of a network structure are epistemological and ergonomic adequacy, as pointed out in Section II-A. Presently, we are not in a position to give a formal and rigorous proof of the epistemological adequacy of our approach. This would require a formal definition of the section of the real world which is to be represented in the knowledge base. An example of such a formal definition is first-order predicate calculus (FOPC), implying the assumption that every important or relevant aspect of the real world can be represented in FOPC. The limitations of FOPC have been discussed in the literature, for example, the exclusion of probabilistic or nonmonotonic reasoning. It has been argued elsewhere, for example, in [12], [44], that semantic networks are equivalent to FOPC. It is pointed out below that we added several extensions to our network structure. For example, the addition of arbitrary procedures allows probabilistic and fuzzy reasoning. So from a formal viewpoint, our definition is more powerful than FOPC. An informal definition of epistemological adequacy was quoted from [22] at the beginning of Section II-A. According to this definition, our semantic network language is epistemologically adequate because we did not have problems in representing the aspects of the world relevant to our applications, as discussed in Section VI.

Besides epistemological adequacy, it is at least convenient to also aim at ergonomic adequacy, a notion introduced in Section II-A. Since this notion does not have a formal definition, only intuitive arguments can be given. For example, we added "sets of modality" to our network structure. A modality set consists of one or more sets of obligatory parts and concretes [see Section III-B2)] of a concept and a set of one or more optional parts and concretes. The concept can be instantiated if one of the sets of obligatory parts and concretes has been instantiated and any subset (including the empty one) of optional parts and concretes. Clearly, the set of modality is not required for reasons of epistemological adequacy, but it is very useful for ergonomic reasons because it allows a more compact and transparent representation of knowledge. Considerations of ergonomic adequacy may recommend a modification of a network structure, even if this structure is epistemologically adequate.

The general data types and the details of the data structures used in ERNEST are given in Figs. 2 and 3, respectively. They are the result of careful considerations of epistemological and ergonomic adequacy. In this section, we only give a condensed overview of these structures, leaving a detailed discussion to the next section. The network consists of three nodes, five links, and nine substructures as shown in Fig. 2. The three nodes are the "concept," the "modified concept," and the "instance." A node is itself a complex data structure defined by a set of 26 slots as shown in Fig. 3 for the concept node. The data structures of the three nodes are identical, except that a pointer to a function in a concept is replaced by the computed value in the corresponding instance; a modified concept is distinguished from a concept only by more restricted ranges for the attribute values. The five links are the "specialization," the "part," the "concrete," the "instance," and the "model" link. The links "specialization," "part," and "instance" are used in most approaches of semantic networks. They correspond to the terms "generalization," "aggregation," and "classification," respectively, in Section II-C. On the contrary, "concrete" and "model" are links introduced in the ERNEST network language to establish relationships between different levels of abstraction, respectively, between model schemes and automatically acquired concepts. Conceptually, the most important and also the intuitively understandable slots are attributes and relations. Each one is itself described by a substructure called "attribute description," "link description," and "relation description." The six other substructures are the "modality description" for the definition of the above-mentioned sets of modality, the "function description" for a standard function definition, the "adjacency" for the representation of time or space relations, the "range" for the description of ranges of attribute values, the "value description" for storing one parameter of the range description (e.g., a lower bound), and the "identification" for distinguishing alternative instantiation paths. Details of these substructures are discussed in the next section. Every substructure in turn is defined by a set of items. A user can work conveniently with the network by means of the software environment described in Section V.

However, definition of the syntax is not yet complete. It was mentioned in the Introduction that it is important to have task-independent control algorithms. In order to achieve this, we impose a set of restrictions on the links. The reasons for those restrictions are discussed in Sections III-B2) and IV. A graphical representation of the restrictions is given in Fig. 4. Basically, the restrictions guarantee that certain cycles in the network are avoided. A consequence is that now there is also a well-defined ordering of nodes along the three links "part," "concrete," and "specialization." These links may be viewed as a three-dimensional coordinate system where every node has a well-defined position which is represented by

```
GENERAL_TYPES: node
                 NODE_TYPES: concept
                             modified concept
                             instance
                             / each node is defined by slots /

               link
                 LINK_TYPES: specialization
                             part
                             concrete
                             model
                             instance
                             / the inverse links are also defined: /
                             specialization of
                             part of
                             concrete of
                             model of
                             instance of

               substrucure
                 SUBSTRUCTURE_TYPE: attribute description
                                    link description
                                    relation description
                                    modality description
                                    value description
                                    function description
                                    adjacency
                                    range
                                    identification
                                    / each substructure is defined by items /
```

Fig. 2. The general data types in the syntactic definition of the network structure ERNEST.

the slot "degree" in a node. The degree of a concept C is a tuple $(d1, d2, d3, d4)$ of integers. The number $d1$ is the length of the longest path leading from some concept K along specialization links to concept C, the number $d2$ is the longest path along concrete-of links (including inherited ones) to C, the number $d3$ is the longest path along part-of links (including inherited ones) to C, and the number $d4$ is the longest path along model-of links to C. The various links are introduced in Section III-B2). It is assumed that every link connecting two concepts has a path length of one unit. Two special classes of concepts are the *minimal concepts* and the *interface concepts*. The minimal concepts have a minimal value of $d2$ and $d3$; to put it differently, they are concepts having no parts and no concretes. The interface concepts have at least one attribute without arguments; for the notion of attributes and arguments, the reader is refered to Section III-B3). Usually, minimal concepts are also interface concepts, but there may also be interface concepts which are not minimal concepts. The interface concepts provide the interface between a purely algorithmic bottom-up phase of processing and a knowledge-based phase alternating between top-down and bottom-up processing; they also are the port for user interaction if such interaction is desirable. Degrees are computed and restrictions are checked automatically by the software environment.

The syntax of the network provides several extensions with respect to other definitions. They are stated briefly in the following eight points. The usefulness of this syntax will become apparent from Sections III-B and III-C. The main extensions in our definition of a semantic net are as follows.

1) The node type "modified concept" allowing the representation of constraints on uninstantiated concepts resulting from computed instances of other concepts.

2) The extension of the definition of the node type "concept" to facilitate automatic acquisition of concepts.

CONCEPT

name of concept	→	text
degrees	→	4 integer
priorities	→	5 integer
information	→	text
model-of	→	link to concepts
model	→	list of links to concept
specialization-of	→	list of links to concepts
specialization	→	list of links to concepts
context-of	→	list of concepts
part-of	→	list of links to concepts
part	→	list of link descriptions
concrete-of	→	list of links to concepts
concrete	→	list of link descripitons
modality	→	list of modality descriptions
attribute	→	list of attribute descriptions
local attribute	→	list of attribute descriptions
analysis parameter	→	list of attribute descriptions
structural relation	→	list of relation descriptions
analysis relation	→	list of relation descriptions
identification	→	list of identifications
judgement	→	function description
instance	→	list of instances
graphic	→	function description
selection of spec.	→	name of function
acquisition rule	→	name of function
frequency	→	1 integer

MODALITY DESCRIPTION

obligatory	→	list of roles
optional	→	list of roles
inherent	→	list of roles
adjacency	→	adjacency

VALUE DESCRIPTION

type of values	→	value type [INTEGER, REAL, ...]
meaning	→	meaning [CENTER, FUZZYPAR, ...]
value	→	value array

ADJACENCY

dimension	→	integer
roles	→	list of roles
diagonal	→	integer vector
matrix	→	bitmatrix
coherent	→	YES or NO

RANGE

kind of range	→	range kind [INTERVAL, FUZZY, ...]
type of values	→	value type [INTEGER, REAL, ...]
complement	→	YES or NO
values	→	list of value descriptions

IDENTIFICATION

path1	→	path of roles or UNIQUE
path2	→	path of roles

ATTRIBUTE DESCRIPTION

role	→	text
modifies	→	YES or NO or DELETED or role
M-modifies	→	NO or DELETED or role
type of values	→	value type [INTEGER, REAL, ...]
restriction	→	range
number of values	→	2x2 integer [min,max/min,max]
comp. of value	→	function description
adj. dependent	→	YES or NO
judgement	→	function description
preference	→	list of ranges
graphic	→	function description
splitting	→	2 integer
comp. of splitt.	→	name of function
comp. of number	→	name of function
comp. of preference	→	name of function
acquisition rule	→	name of function
frequency	→	1 integer

LINK DESCRIPTION

role	→	text
modifies	→	YES or NO or DELETED or role
M-modifies	→	NO or DELETED or role
goal node	→	list of concepts [xor list]
type of node	→	node type [CONCEPT, ...]
context depending	→	YES or NO
number of links	→	2 integer [min,max]
transformation	→	list of value descriptions
judgement	→	function description
preference	→	list of ranges
precedence	→	integer
complement	→	YES or NO
splitting	→	2 integer
comp. of splitt.	→	name of function
comp. of number	→	name of function
acquisition rule	→	name of function
frequency	→	1 integer

RELATION DESCRIPTION

role	→	text
modifies	→	YES or NO or DELETED or role
M-modifies	→	NO or DELETED or role
judgement	→	function description
adj. dependent	→	YES or NO
splitting	→	2 integer
comp. of split.	→	name of function
acquisition rule	→	name of function
frequency	→	1 integer

FUNCTION DESCRIPTION

name	→	name of function
argument	→	list of arguments [{role.}role]
inverted function	→	name of function
test of arguments	→	name of function
fusion of arguments	→	name of function

Fig. 3. ERNEST data structures. The slots relevant only for knowledge acquisition are marked as slanted. As mentioned in the text, knowledge acquisition is not a topic of this paper.

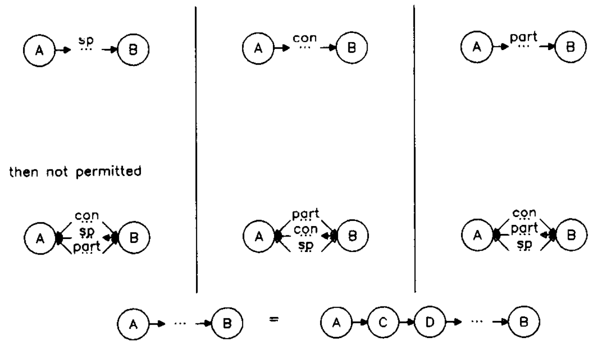


Fig. 4. The restrictions on the links. The left, middle, and right parts present independent alternative restrictions. For example, the left part means that if a concept *B* can be reached from *A* by a sequence of specialization links, then there must not be specialization links from *B* to *A*, nor concrete links from either *A* to *B* or *B* to *A*, nor part links from either *A* to *B* or *B* to *A*.

3) The link type "concrete" allowing the representation of different conceptional systems.

4) The link type "model" allowing the representation of the relation between *a priori* knowledge and automatically acquired knowledge.

5) The distinction between "context-dependent" and "context-independent" parts allowing the representation of context-sensitive relations.

6) The introduction of "sets of modality" allowing the efficient representation of different object descriptions in one concept.

7) The "adjacency matrix" allowing the efficient representation of time and space constraints.

8) The inclusion of a standard function definition facilitating constraint propagation by modified concepts.

These extensions of the syntax are supplemented by two points concerning the use of the networks.

1) The formulation of six task-independent rules precisely defining the computation of modified concepts and instances.

2) The combination of these rules with graph search algorithms to handle the control problem.

B. The Semantics of the Network

1) Types of Nodes: In this section, the interpretation of the different types building up an ERNEST network is given. The point of view with which we will look at the network is: how is a certain section of the real world and how are certain aspects of this section modeled in the knowledge representation language of an automatic system? As mentioned in the last section, three types of nodes are distinguished, which are the concept, modified concept, and instance node.

A basic requirement for any image or speech understanding system is the ability to represent classes of objects, events, or abstract conceptions having certain common properties. This is done in ERNEST (and in other approaches to semantic nets) by a special node type, called a *concept*. The special case that a concept represents precisely one element is not excluded. In the context of image or speech understanding, an important step is to associate certain intervals of the sensor data with certain concepts in the knowledge base. For example, some subset of pixels in a TV image is recognized as a truck, where TRUCK is a concept in the knowledge base. This particular manifestation of a member of the concept TRUCK is called an instance and is represented by another node type, the *instance* node. A concept and an instance of it are related by an *instance link*. This was mentioned as the "classification axis" in Section II-B.

In addition to the concept and instance node, a third node type is provided, the modified concept. The basic activity of an image or speech understanding system is the computation of instances which are consistent with the stored knowledge and the observed sensor data. In an intermediate state of processing, it may occur that instances of some concepts already have been computed, but that instances of some other concepts cannot yet be computed because certain prerequisites are missing. A precise definition of those prerequisites is given in Section III-C below. Nevertheless, the information available from the instances may be used to constrain or to modify the uninstantiated concepts; this results in the *modified concepts*. For example, the values of certain attributes of an instance may be used to constrain the range of values of certain attributes of an uninstantiated concept. This in turn results in a more constrained knowledge base and in a reduction of the complexity of the instantiation process. An example of constraining the value of an attribute is given in Section III-B3).

According to classification, *instances* are associated with *concepts*. The connection among instance nodes, concept nodes, and real world objects is illustrated in Fig. 5. Each instance establishes a connection between exactly one concept and a unique collection of real world objects

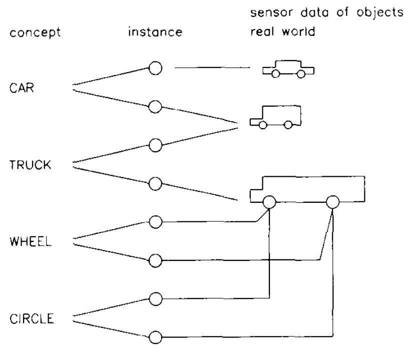


Fig. 5. Potential relationships among concepts, instances, and real world objects.

or events (to be more precise, sensor data resulting from real world objects and events). A collection of real world objects or events can be connected to more than one concept via instances and vice versa. Notice that there are connections between one real world object and different concepts which are acceptable, and other ones which establish a contradiction. In Fig. 5, for example, the CIRCLE and the WHEEL classifications for identical objects are correct. But only one of the connections between the middle object and the concepts CAR, respectively, TRUCK, is admissible. A concept in the ERNEST network is a model (i.e., a description) of a class of objects or events, but not a prototype (i.e., not an example or a typical representative of the class). Therefore, such a concept may be restricted, dependent on different situations in an analysis process. For example, if the x, y position of a wheel is known for the truck in Fig. 5, the *a priori* free positions of the other wheels then are restricted with respect to the detected one. Although no instance is built up for a second wheel, the situation has changed. The concept WHEEL in the context "second wheel of a truck with one wheel detected" can be restricted. We say a *modified concept*, as mentioned above, can be built up.

2) Types of Links: With the interpretation of the three node types, the link types *instance* and *instance-of* also are fixed according to the links between concepts and instances in Fig. 5.

In the following, the example of a truck is used to illustrate the realization of a network and the meaning of the different structures. Fig. 6(a) shows a rough model of a truck, and Fig. 6(b) the related network. Fig. 7 shows the textual representation of the concepts TRUCK and DRIVERS_CAB.

Besides the instance relationship, four organizational axes are distinguished. All of them define a partial order on the set of concepts. It is common practice to define a concept with reference to other concepts, as introduced already in Section II-B. These references are made explicit by special types of links. From Fig. 2, it is evident that the inverse links are also provided, although this will not be mentioned in the following. A concept K may be defined as a refinement of a more general concept A by

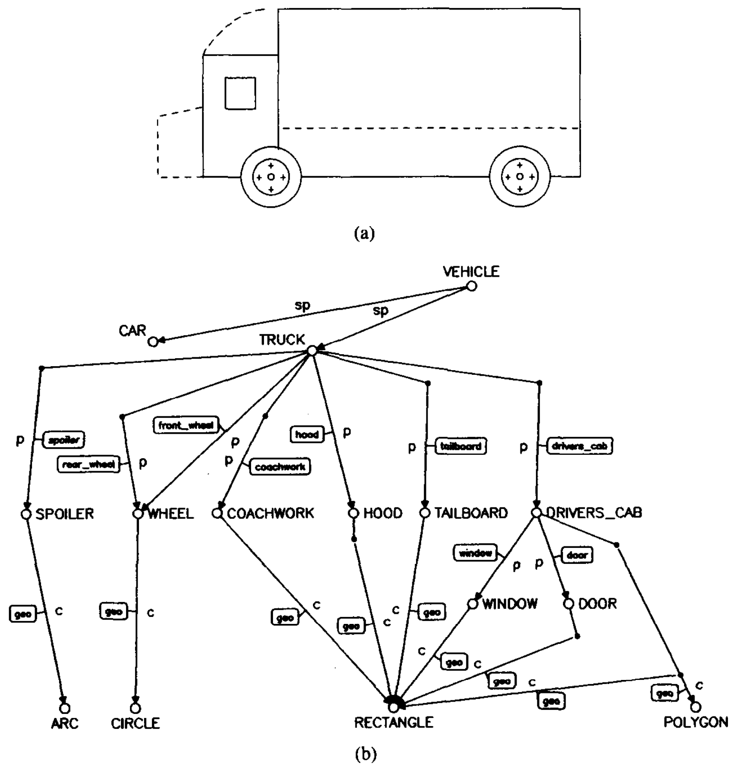


Fig. 6. (a) A simple graphic representation of a truck. (b) A network of the model TRUCK.

stating the additional and/or modified properties of *K*, and by assuming that otherwise, *K inherits* the properties of *A*. By "properties," we summarize the slots part, concrete, attribute, analysis parameter, structural relation, and analysis relation in the data structure of a concept node. The relation between the general concept *A* and the specialized concept *K* is indicated by a *specialization* link from *A* to *K*. A modification of properties is represented in the item "modifies" of the corresponding "attribute," "relation," and/or "link description" in Fig. 3. In our example, the concept TRUCK is a specialization of the concept VEHICLE as shown in Fig. 6(b).

In addition, a concept may be composed of certain parts or components $U1$, $U2$, $\cdots$, UN. For example, in Figs. 6(b), 7(a), parts of the TRUCK are the SPOILER, the FRONT_WHEEL, the REAR_WHEEL, and so on. The relation between a concept *K* and one of its parts *U* is represented by a *part link* from *K* to *U*. From Figs. 6(b), 7(a), it can be seen that FRONT_WHEEL and REAR_WHEEL are just particular "roles" within the concept WHEEL. In general, the values of a part may be defined by one or more concepts, and a part may be given a particular role. Another example is the part WINDOW in Fig. 7(b) having the role WINDOW defined by the concept WINDOW. In ERNEST, a part or a concrete is defined in the substructure *link description*. The link description in Fig. 3 contains an item "goal node" to represent the defining concepts and an item "role" to represent the particular role of a part.

In image or speech understanding, it often occurs that a certain part can only be recognized in the context of the corresponding object having this part. For example, in Fig. 6(a), the window of the truck is seen as a rectangle. This rectangle obtains its meaning only in the context of the whole image. Therefore, the concept WINDOW in Fig. 7(b) is defined as a *context-dependent part* of the DRIVERS_CAB. In the ERNEST representation of this situation, the slot "context of" of the concept WINDOW has the entry "DRIVERS_CAB." In the concept DRIVERS_CAB, the slot "part" contains a list of link descriptions, one of which has the role WINDOW. The item "context depending" of this link description has entry "YES" [see also Fig. 7(b)]. On the other hand, there may be parts which can be recognized individually. An example is the front wheel in Fig. 6(a). Therefore, the front wheel is a *context-independent part* of the TRUCK. In Fig. 7(a), an explicit indication of context independence is omitted. In the ERNEST representation the corresponding slot, "context of" would have entry "NIL," and the corresponding item "context depending" would have entry "NO." It should be noted that context-dependent and context-independent parts are treated differently during instantiation of concepts; this point is discussed in Section III-C.

In order to motivate some of the formal restrictions in Fig. 4 and the link type *concrete*, the description of aggregation in [26] is reported: "for example, the parts of John Smith, viewed as a physical object, are his head,

```
TRUCK

  PART: spoiler                              PART: coachwork
    GOAL_NODE:       SPOILER                   GOAL_NODE:       COACHWORK
    CONTEXT_DEP:     NO                        CONTEXT_DEP:     NO
  PART: front_wheel                          PART: hood
    GOAL_NODE:       WHEEL                     GOAL_NODE:       HOOD
    CONTEXT_DEP:     NO                        CONTEXT_DEP:     NO
    JUDGEMENT:       restr_radius            PART: tailboard
      INV_FUNCT:       inv_restr_radius        GOAL_NODE:       TAILBOARD
  PART: rear_wheel                             CONTEXT_DEP:     NO
    GOAL_NODE:       WHEEL                   PART: drivers_cab
    CONTEXT_DEP:     NO                        GOAL_NODE:       DRIVERS_CAB
    JUDGEMENT:       restr_radius              CONTEXT_DEP:     NO
      INV_FUNCT:       inv_restr_radius

  MODALITY:
    OBLIGATORY:      front_wheel, rear_wheel, coachwork, drivers_cab
    OPTIONAL:        spoiler
    INHERENT:        ---
  MODALITY:
    OBLIGATORY:      front_wheel, rear_wheel, tailboard, hood, drivers_cab
    OPTIONAL:        spoiler
    INHERENT:        ---

  ATTRIBUT: height
    TYPE_OF_VAL:     REAL
    NUMB_OF_VAL:     1
    RESTRICTION:     [2.0 - 3.5]
    COMP_OF_VAL:     compute_height
      ARGUMENT:        front_wheel.radius, rear_wheel.radius, coachwork.height
                       hood.height, tailboard.height
      INV_FUNCT        inv_compute_height
  ATTRIBUT: length
    TYPE_OF_VAL:     REAL
    NUMB_OF_VAL:     1
    RESTRICTION:     [6.0 - 12.0]
    COMP_OF_VAL:     compute_length
      ARGUMENT:        drivers_cab.length, hood.length, coachwork.length
      INV_FUNCT        inv_compute_length

  STRUCTURAL_RELATION: height_smaller_length
    JUDGEMENT:       judge_h_s_l
      ARGUMENT:        height, length
      INV_FUNCT:       inv_judge_h_s_l
  ANALYSIS_RELATION: wheels_down
    JUDGEMENT:       judge_w_d
      ARGUMENT:        front_wheel.centre, rear_wheel.centre, coachwork.angles
                       tailboard.angles, drivers_cab.angles
      INV_FUNCT:       inv_judge_w_d

  JUDGEMENT: judge_truck
    ARGUMENT:        spoiler, front_wheel, rear_wheel, coachwork
                     hood, tailboard, drivers_cab, height, length
                     height_smaller_length, wheels_down
```

(a)

```
DRIVERS_CAB

  PART: window
    GOAL_NODE:       WINDOW
    CONTEXT_DEP:     YES
  PART: door
    GOAL_NODE:       DOOR
    CONTEXT_DEP:     YES
  CONCRETE: geometry
    GOAL_NODE:       RECTANGLE, POLYGON

  MODALITAET:
    OBLIGATORY:      window, door, geometry
    OPTIONAL:        ---
    INHERENT:        door

  ATTRIBUT: height
    TYPE_OF_VAL:     REAL
    NUMB_OF_VAL:     1
    RESTRICTION:     [1.5 - 3.0]
    COMP_OF_VAL:     compute_height
      ARGUMENT:        angles
      INV_FUNCT        inv_compute_height
  ATTRIBUT: length
    TYPE_OF_VAL:     REAL
    NUMB_OF_VAL:     1
    RESTRICTION:     [0.75 - 3.0]
    COMP_OF_VAL:     compute_length
      ARGUMENT:        angles
      INV_FUNCT        inv_compute_length
  ANALYSIS_PARAMETER: angles
    TYPE_OF_VAL:     RECORD, point
    NUMB_OF_VAL:     12
    RESTRICTION:     ---
    COMP_OF_VAL:     compute_angles
      ARGUMENT:        door.angles, window.angles, geometry.angles
      INV_FUNCT        inv_compute_angles

  STRUCTURAL_RELATION: height_greater_length
    JUDGEMENT:       judge_h_g_l
      ARGUMENT:        height, length
      INV_FUNCT:       inv_judge_h_g_l
  ANALYSIS_RELATION: window_angles_inside
    JUDGEMENT:       judge_w_a_i
      ARGUMENT:        angles
      INV_FUNCT:       inv_judge_w_a_i

  JUDGEMENT: judge_drivers_cab
    ARGUMENT:        window, door, geometry, height, length, angles
                     height_greater_length, window_angles_inside
```

(b)

Fig. 7. (a) The concept TRUCK. (b) The concept DRIVERS_CAB.

arms, etc. When viewing as a social object, they are its address, social insurance number, etc." Two "worlds" or *conceptional systems* are distinguished in this example. A concept modeling a person has different parts within each of these systems. Parts in the social system are social conceptions; parts in the physical system are physical conceptions. In complex applications, more than one such conceptional system will occur, e.g., in image understanding, conceptional systems like geometry, named objects, or motions will occur. Conceptional systems like syntax, semantics, and pragmatics build up the knowledge base for speech understanding tasks. In the ERNEST network, part and specialization relationships are restricted in the way that they are only allowed inside the same conceptional system. For analysis purposes, conceptual systems must be ordered in the sense that a hierarchy of levels of abstraction is established. Each level corresponds to one conceptional system and vice versa. For the examples mentioned above, the organizations

image—lines—geometry—named objects—motions

and

speech—signal—phonemes—words—syntax—semantics—pragmatics—dialogue

may build up sequences of such levels with increasing abstraction. Relationships between concepts belonging to different levels of abstraction are established by the *concrete link*. Each level of abstraction consists of concepts having identical degree with respect to the concrete links; compare Fig. 4 and Section III-A. The membership of concepts to different conceptional systems also influences the compatibility of corresponding instances. Because the concepts WHEEL and CIRCLE in Fig. 5 belong to different systems, instances connecting both concepts to identical real world objects can all be correct. On the contrary, because TRUCK and CAR are within the same conceptional system and are not in part relationship, instances connecting both to an identical object compete with each other. See also Fig. 6(b) for the distinction between parts and concretes.

In the definition of a concept, there may be parts and/or concretes (including inherited ones) which are obligatory and others which are optional. A set of obligatory parts and concretes together with the associated set of optional parts and concretes is called a *modality set*. In order to increase the compactness of a knowledge base, we allow that a concept is defined by several modality sets; each individual modality set is sufficient to compute an instance. In ERNEST, a modality set is defined by the substructure *modality description*, and the modality descriptions are attached to the slot ''modality'' of the defined concept. From Fig. 3, it can be seen that a modality description has items ''inherent'' and ''adjacency'' in addition to the obligatory and optional elements. Inherent parts and concretes are those which can be inferred from the instantiation of a concept, but which are not manifest in the sensor data. For example, when seeing a truck (or after computing an instance of the concept TRUCK), one may usually assume that it has an engine, although this will not be visible under standard viewing conditions. The concept TRUCK has two modality descriptions. In our example, a truck must have front wheels, rear wheels, a driver's cab, and either a coachwork or a tailboard and a hood. In both cases, a spoiler is optional. If necessary and appropriate, the item ''adjacency'' allows one to impose a certain temporal (or spatial) order on parts and concretes. This temporal order is defined in the substructure *adjacency* by means of a bit matrix. For example, assume that the modality description contains only the concepts A, B, and C, that A must precede B, and that C may precede B, but not A. A bit matrix then has rows A, B, C and columns A, B, C. An entry ''1'' in row i and column j indicates that part i must precede part j in time, and so on. The item ''coherent'' indicates whether parts have to be spatially or temporally adjacent. For example, the sky is above the meadow in an image under standard viewing conditions, but there may be something in between (e.g., a mountain range); therefore, sky and meadow are not coherent. On the other hand, a roof is above a gable, and they are coherent for a standard house.

The relationships specialization, part, and concrete build up the three-dimensional hierarchy of an ERNEST knowledge base. They provide the means for well-structured representation of knowledge. The instance link connects concepts in the knowledge base to instances computed from sensor data. In order to facilitate automatic knowledge acquisition, a fifth link, the *model link*, was introduced. In this paper, however, we concentrate on knowledge representation and utilization. Therefore, no further explanation is given about knowledge acquisition in ERNEST. More information can be found in [43].

The above discussion introduced the three node types (concept, modified concept, and instance) and the five link types (specialization, part, concrete, instance, and model). In comparison to other network definitions, the concept and the instance node are standard elements of a semantic network and are necessary for reasons of epistemological adequacy. The modified concept was introduced only to increase the efficiency of knowledge utilization. The specialization, part, and instance link are also standard elements in other versions of semantic networks and are necessary for reasons of epistemological adequacy. The distinction between context-dependent and independent parts is necessary in our applications. The concrete link is epistemologically necessary, while the model link is introduced for reasons of ergonomic adequacy. The modality sets enhance the compactness of a knowledge base.

3) Attributes, Relations, and Other Substructures: From Fig. 3, it is apparent that a concept node is an elaborate data structure defined by several slots. Some of them are self-explaining (e.g., ''name of concept''), some have been introduced above (e.g., ''degree'' or ''model''), and the remaining ones will be introduced in the following.

A physical object or an event usually has certain *attributes* which are physical quantities, for example, size, weight, color, or prize. In ERNEST, an arbitrary number of attributes may be attached to the slot ''attribute'' in the corresponding concept. In addition, we distinguish the *local attribute* which is not transferred to specializations of the concept. An attribute is transferred to specializations unless this is excluded explicitly in the item ''modifies'' of the substructure ''attribute description.'' Apparently, the local attribute is not necessary because it also can be replaced by modification of attributes, but it is convenient to have this option. Furthermore, we provide *analysis parameters* which are not attributes of an object and do not contribute to the intensional description, but they are parameters required for pattern analysis. Typical examples are the frame length and repetition rate in speech understanding or the number of images in an image sequence. Attributes, local attributes, and analysis parameters are defined by the substructure *attribute description*. It seems unnecessary to discuss every item of the attribute description. The main items are the ''role,'' the ''type of values,'' the ''restriction,'' the ''number of values,'' and the ''computation of value.'' The ''role'' gives the functional role of the attribute. The ''type of values'' defines the general type of the attribute values. In ERNEST, the types Boolean, Character, Integer, Real, Set, Tree, and Record are allowed. The ''restriction'' specifies the allowed or expected set of values via ''range'' and ''value description.'' An example is the attribute ''height'' in Fig. 7(a), a real-valued attribute restricted to 2.0–3.5 units of length. Thus, type and restriction define the range of values of an attribute. The ''number of values'' defines the required number of attribute values, for example, the number of elements of a vector or matrix. The item ''computation of value'' contains just a pointer to a function description referencing a function which can compute an actual value of the attribute given the sensor data. This function needs certain arguments. The function definition and the arguments are treated below. There may be special attributes referencing a function having no arguments, for example, a function reading a value from a file. This occurs only in the "interface concepts" introduced in Section III-A. An attribute of this type provides the means for transferring results of initial segmentation to the knowledge base.

Parts or concretes of a concept may have to be arranged in a certain spatial and/or temporal order. For example, the positions of the wheels of a truck form a rectangle, or a traffic light should become green before a car starts moving. Certain relationships between parts and/or concretes of a concept are listed in the slot *structural relation* of this concept. Every element in a structural relation is defined by a substructure *relation description*. The main items in this substructure are the "role" of a particular relation and its "judgment," which is a pointer to a function testing the relation. The value returned by the function measures the degree of fulfillment of the relation. If the temporal relation is only a time sequence of objects or events, an alternative representation in ERNEST is the above-mentioned modality set combined with the adjacency substructure. Although this alternative is not necessary for reasons of epistemological adequacy, it is useful for reasons of efficient implementation of time or space adjacency relations. Similar to the distinction between attributes and analysis parameters, we distinguish between structural relations and analysis relations. An example is a relation among the number of pixels, depth, and object size.

The original sample values of images or speech usually are corrupted by noise; the results of initial segmentation are far from being perfect. Therefore, an instance of a concept may be more or less erroneous. The slot *judgment* of a concept contains a pointer to a function description computing a "judgment" of an instance. We avoid terms like score or certainty because a judgment in ERNEST may be a tuple of real numbers measuring the quality, certainty, and static and dynamic priority of an instance. The quality measures the average value of a result, for example, the expected probability of correctly recognizing a word or a line of a certain length. The certainty measures the individual reliability of an instance, for example, of a particular word or a particular line. The static priority measures the closeness to primitive concepts (important in top-down processing) and to goal concepts (important in bottom-up processing). The dynamic priority measures the ambiguity of an intermediate result with respect to task-specific knowledge. In general, the certainty CF of an instance is based on a function of the type

$$\begin{aligned}\mathrm{CF} = g(&\text{certainties of parts, certainties of concretes,}\\ &\text{certainties of attributes, certainties of relations}).\end{aligned}$$

Arguments of this function are the judgments of the parts, concretes, link descriptions, attribute descriptions, and of both kinds of relations. All arguments are referred by the role of the corresponding substructure. The scheme to judge an instance is not fixed in ERNEST. Depending on the application, fuzzy logic, distance measurements, or probabilities are used.

It was mentioned frequently that functions can be attached to slots or items of concepts or substructures, respectively. In order to enhance standarization of attached functions, the substructures *function description* and *value description* are provided as a basis of procedural attachment. A function definition in the ERNEST network includes the explicit notation of the "arguments," and it is possible to also refer the inverse. It should be mentioned that it is not possible, but also not necessary to refer the inversions of all functions in a knowledge base. However, our experiences with knowledge-based systems showed that a lot of functions calculating attributes or relations are very simple, e.g., attributes may be defined by the sum or product of other ones. An example for relations which is also very simple is that different parts of an object are not allowed to cover overlapping image areas and should be neighbored in some sense. The inversion is quite simple. Given one part, the area of the other ones is restricted to the complement in the image, and the required neighborhood can be expressed by marking a few pixels. This simple restriction is able to reduce the detection of other parts very powerfully because only a small subset of the image must be analyzed. Relevant functions are those for computing the "judgment" (in a concept, or an attribute, link, or relation description), and the "computation of value" (in an attribute description). All of these functions have to be defined in an ERNEST network by the substructure "function description"; see Fig. 3. Here, the arguments of the function, the name of the function, and the name of the inverse of the function with respect to each argument are denoted. By the definition of the network formalism, the set of potential arguments is restricted for each of the functions mentioned above. In order to identify arguments, we use the "roles" which are defined uniquely by each of the relevant substructures of a concept. Those are the "link description," the "attribute description," and the "relation description." For the different functions, the following roles can be used to represent arguments.

1) For "judgment" of a concept: roles of links, attributes, and relations, which are defined for the same concept.

2) For "judgment" in the link descriptions and attribute descriptions: no explicit argument can be noted.

3) For "computation of value": pairs SELF. r_1 where r_1 is the role of an attribute which is defined for the same concept; pairs r_1. r_2 where r_1 is the role of a link description in the same concept, and r_2 is the role of an attribute in one of those concepts which are referred in the "goal node" of the link description with role r_1; pairs SUPER. r_2 where r_2 is the role of an attribute in one of those concepts which are referred in the slot "context of" in the same concept.

4) For "judgment" in the relation description: pairs SELF. r_1 and pairs r_1. r_2 as above.

While the activation of the function itself results in concrete values if all arguments are known, the inverse is able to further constrain the "restriction" of attributes in modified concepts. In a similar way, the values of an attribute may be restricted, if not all of the argument values are known (see Section III-C). As an example of restricting the values of an attribute, consider a concept PIECE_OF_LAND having parts SIDE_1 and SIDE_2. The attribute "a" (area) of PIECE_OF_LAND is computed by $a = lw$ where "l" and "w" are the attributes length of side 1 and length of side 2, respectively. Assume that in the three attributes descriptions, the item "restriction" imposes the constraints $400\ \mathrm{m}^2 \le a \le 1000\ \mathrm{m}^2$, $8\ \mathrm{m} \le w \le 20\ \mathrm{m}$, and $40\ \mathrm{m} \le l \le 100\ \mathrm{m}$. Having an instance of SIDE_2 giving $l = 80$ m allows one to compute (bottom up) a modified concept of PIECE_OF_LAND with $640\ \mathrm{m}^2 \le a \le 1000\ \mathrm{m}^2$. From this modified concept, one may compute (top down) a modified concept of SIDE_1 with $8\ \mathrm{m} \le w \le 12.5$ m. This step also requires that the "inverse function" $w = a/l$ can be referenced. The substructure "function description" contains an item "inverted function" which gives the facility to also reference the inverse function.

To also allow a graphical representation of the content of concepts (e.g., wire frame) or attributes (e.g., a contour), a "graphic" slot is provided. It references a function generating the graphic representation.

In order to allow some flexibility in the design of a concept representing a certain conception, it is useful to distinguish different constraints in the instantiation of a concept. For example, assume that a chair has among its parts four legs. One possibility of modeling this is to actually attach four different parts LEG to the concept CHAIR. If all four legs are geometrically (almost) identical, the concepts representing the parts are (almost) identical. In this case, it may be convenient to attach only one part LEG to CHAIR, but indicating in the item "number of links" in the link description that four (different) instances of LEG are necessary. Representing a triangle can be done by a concept TRIANGLE having a part SIDE, and the SIDE has a part EDGE. In this case three different instances of both SIDE and EDGE are necessary. An instance of SIDE needs two instances of EDGE, so it must be possible to indicate that, for example, two instances of EDGE have one instance of SIDE in common. Again, an alternative way of representing a triangle would be to represent three different parts SIDE and three different parts EDGE. Our intention is to leave the choice of the model to the designer, and not to enforce a certain representation by the network realization. Another example is the representation of a heart cycle by a concept CYCLE having the three parts CONTRACTION, STAGNATION, and EXPANSION, which in turn all refer to the moving object HEART. In this case, for one instance of CYCLE, instances of CONTRACTION, STAGNATION, and EXPANSION are necessary, all of which need one and the same instance of HEART. In ERNEST, the different constraints on instantiation are defined by the substructure *identification* which is refered to in the slot "identification" of the corresponding concept. The three types of constraints introduced by the above examples are summarized in Fig. 8. In the first case, there are no entries in the slot identification of the concepts A and B. Via two links, the concept F is referred twice by A and once by B. That is the conceptual point of view. If one associates, e.g., the concept A with "car," B with "airplane," and F with "window," it is evident that one instance of "window" is not sufficient to fill the properties of "window of car" and "window of airplane." Therefore, different instances of F are necessary. For each binding of F in Fig. 8, one instance is necessary. Associate, for the second case, A with "contraction of a heart," B with "expansion of a heart," and F with "heart"; then only one instance for F is necessary, as for the instantiation of A and B, the same heart has to be used. Therefore, the concept F is marked unique to guarantee this. For the third case, associate A with "passing of two cars" C, respectively D, with one "car" and F with "street"; then by the identification path, one can mark that C and D must be on the same street if they pass. Another car (concept E) naturally can drive on another street in the picture.

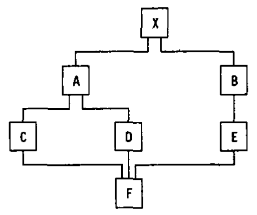


Fig. 8. Specification of alternative constraints on the instantiation of X. In the notation below, $\mathrm{in}_i(K)$ denotes the ith instance of a concept K. All links in the above figure are part links, the same constraints may be specified for concrete links. Case 1: No entry in the slot "identification" of concepts A and B (this is the default situation). Computed instances: $\mathrm{in}(X)\ \{\mathrm{in}(A)\ [\mathrm{in}(C)\ \mathrm{in}_1(F)]\ [\mathrm{in}(D)\ \mathrm{in}_2(F)]\ \mathrm{in}(B)\ [\mathrm{in}(E)\ \mathrm{in}_3(F)\}$. Case 2: Pointer to substructure identification; entry UNIQUE in item "path 1" of concept F. Computed instances: $\mathrm{in}(X)\ \{\mathrm{in}(A)\ [\mathrm{in}(C)\ \mathrm{in}(F)]\ [\mathrm{in}(D)\ \mathrm{in}(F)]\ \mathrm{in}(B)\ [\mathrm{in}(E)\ \mathrm{in}(F)\}$. Case 3: Pointer to substructure identification; entry in item "path 1" of concept A: C, F, entry in item "path 2" of concept A: D, F. $\mathrm{in}(X)\ \{\mathrm{in}(A)\ [\mathrm{in}(C)\ \mathrm{in}_1(F)]\ [\mathrm{in}(D)\ \mathrm{in}_1(F)]\ \mathrm{in}(B)\ [\mathrm{in}(E)\ \mathrm{in}_2(F)]\}$.

C. The Pragmatics of the Network

Besides the epistemological and ergonomic adequacy of a definition of a semantic network, another important aspect is the utilization of this network for image or speech understanding. As mentioned above, the main activity in the network is the computation of instances out of concepts given certain sensor data. Computation of instances only depends on the syntax of the network; in particular, it depends on the three organizational hierarchies of specialization, part, and concrete, on the distinction between context-dependent and independent parts, on the introduction of modified concepts and sets of modality, and on the definition of certain paths of instantiation via the identification substructure. It turns out that six rules are sufficient to define the instantiation process. These rules complete the definition of the network by defining the pragmatics of the formalism in the sense of the procedural semantics [17]. Different from PSN, the rules for building up instances are defined globally for the whole network, without respect to a task domain. This is possible because the rules only make use of the syntax of the network language, not of the semantics or the meaning of concepts in the network. The six rules are the basis for problem-independent control as discussed in Section IV.

In the following, the rules will be illustrated by the formal example in Fig. 9 and by the network in Fig. 6(b) which gives a simple example from image analysis. It is assumed that the goal concept for the instantiation is the concept TRUCK with respect to the first modality set [see Fig. 7(a)]. It is further assumed in Fig. 10 that we consider the situation where the hth instance in_h *(RECTANGLE)* [see Fig. 11(a)] of the concept RECTANGLE (a concrete of the concept DRIVERS_CAB) has been computed so far.

In order to compute an instance of a concept A in Fig. 9, there must be instances of all of its concretes and parts, including those inherited from more general concepts. Instances of parts and concretes are only necessary if they belong to the obligatory parts and concretes of some set of modality. Furthermore, requiring an instance of a part is only possible if it is a context-independent part. If a concept A is a context-dependent part of some concepts X and Y, there must be at least one instance of either X or Y when instantiating A. Obviously, computation of instances mainly proceeds bottom up. This causes the problem that computation of an instance of A having a context-dependent part M and context-independent parts L and N requires instances of L and N and also of M. But computing an instance of M requires an instance of A. The problem is handled by first computing a *partial instance* of A by requiring only instances of the context-independent parts. This is summarized in Rule 1 below. Having the partial instance of A, an instance of the context-dependent part M can be computed. Having an instance of M, the partial instance A can be completed. This process is summarized by Rule 2. Of course, a recursive application of Rule 1 may be necessary. In Fig. 9, the concept A is itself a context-dependent part of X and Y. So at first, a partial instance of X (or of Y) has to be constructed by Rule 1 before a partial instance of A can be constructed. The first rule is as follows.

Rule 1:

IF for a *concept A* or a *modified concept* $mod_j(A)$ with respect to one obligatory set of a modality of A, instances for those concepts exist, which are referred to by the following slots in A or slots inherited to A without modification:

- *concrete* **AND**
- *part*, if the item *context depending* in the link description is equal to **NO AND**
- one partial instance of one concept referred in *context-of*, if this slot is not equal to **NIL**

THEN build up *partial instances* $inp_k(A)$ as follows:

- construct an empty instance of A

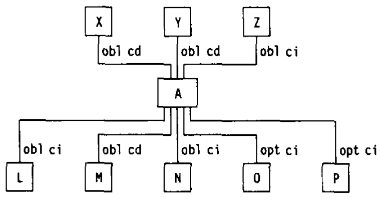


Fig. 9. An example for use of Rule 1; the abbreviations obl, opt, cd, and ci stand for obligatory, optional, context dependent, and context independent, respectively. All links are part links from the upper to the lower concept; for example, concept A is an obligatory context-dependent part of Y, and A has P as an optional context-independent part.

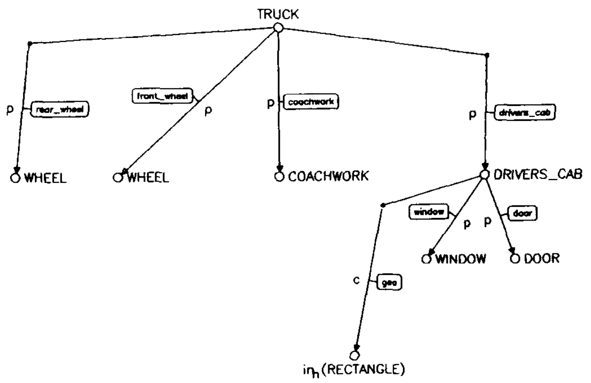


Fig. 10. An example of an analysis process.

- connect the instance with those referred to by the premise
- activate the attached functions in the sequence: judgments of links, calculation of attributes, judgments of attributes, judgment of relations, judgment of the concept.

Because of context-dependent parts and optional links, values do not exist for all arguments of the activated functions [e.g., the attribute "angles" in Fig. 7(b)]. Nevertheless, the knowledge of existing values of arguments and the function itself can be used if the following strategy is applied. The restriction values are also transferred to the functions. The functions themselves decide whether the existing values and the restrictions are sufficient for the estimation of results. In the case of attributes, this estimation is a new and tighter restriction. For the other cases, which are all judgments, the estimation must be optimistic.

Rule 2:

IF a *partial instance* $inp_j(A)$ of a concept A exists **AND** instances for all those concepts exist, which are referred to as *part* with item *context depending* equal to **YES** in the link description and are members of the *obligatory* set which was used for the construction of $inp_j(A)$

THEN build up *new instances* $in_k(A)$ out of $inp_j(A)$ as described above.

Since Rules 1 and 2 only consider obligatory parts and concretes of a set of modality, Rule 3 checks whether there are instances of optional parts or concretes. If this is the case, an instance is extended by these optional compo-

```
inh(RECTANGLE)

 ATTRIBUTE: angles
   RESTRICTION     ---
   VALUE           [ (30,75) (70,75) (70,138) (30,138) ]

 JUDGEMENT: 0.93
```

(a)

```
inpi(DRIVERS_CAB)

 CONCRETE: geometry
   GOAL_NODE       inh(RECTANGLE)

 ATTRIBUTE: height
   RESTRICTION     [2.9 - 2.9]
   VALUE           2.9
 ATTRIBUTE: length
   RESTRICTION     [1.7 - 1.7]
   VALUE           1.7
 ANALYSIS_PARAMETER: angles
   RESTRICTION     [ (x,x) (x,x) (x,x) (x,x)
                     (x,x) (x,x) (x,x) (x,x)
                     (30,75) (70,75) (70,138) (30,138) ]
   VALUE           ---

 STRUCTURAL_RELATION: height_greater_length
   VALUE           0.96
 ANALYSIS_RELATION: window_angles_inside
   VALUE           1.0 (optimistic)

 JUDGEMENT: 0.98 (optimistic)
```

(b)

```
inj(DRIVERS_CAB)

 PART: window
   GOAL_NODE       ink(WINDOW)
 PART: door
   GOAL_NODE       inl(DOOR)
 CONCRETE: geometry
   GOAL_NODE       inh(RECTANGLE)

 ATTRIBUTE: height
   RESTRICTION     [2.9 - 2.9]
   VALUE           2.9
 ATTRIBUTE: length
   RESTRICTION     [1.7 - 1.7]
   VALUE           1.7
 ANALYSIS_PARAMETER: angles
   RESTRICTION     ---
   VALUE           [ (38,85) (62,85) (62,130) (38,130)
                     (40,110) (60,110) (60,125) (40,125)
                     (30,75) (70,75) (70,138) (30,138) ]

 STRUCTURAL_RELATION: height_greater_length
   VALUE           0.96
 ANALYSIS_RELATION: window_angles_inside
   VALUE           0.95

 JUDGEMENT: 0.83
```

(c)

Fig. 11. (a) The instance in_h (RECTANGLE). (b) The partial instance inp_i(DRIVERS_CAB). (c) The instance in_j(DRIVERS_CAB).

nents. For example, if there is an instance of 0 but not of P in Fig. 9, the instance of A is extended by in(0).

Rule 3:

IF an *instance* $in_j(A)$ of a concept A exists **AND** at least one instance of a concept exists, which is *optional* due to the modality used for constructing $\text{in}_j(A)$

THEN build up *extended instances* $in_k(A)$ out of $\text{in}_j(A)$.

In the situation of Fig. 10, the premise of Rule 1 is satisfied for the concept DRIVERS_CAB [see Fig. 7(b)] because both part links are context dependent. The result is the partial instance inp_i(DRIVERS_CAB) which is shown in Fig. 11(b). By not knowing all arguments of the analysis parameter "angles," only the "restriction" could be further restricted. After the instantiation of the context-dependent concepts WINDOW and DOOR, the second rule can be applied. This results in the instance in_j(DRIVERS_CAB) [see Fig. 11(c)] where all attributes are calculated and all judgments are done.

If a goal concept for an analysis process is known, recursive application of these three rules results in a search tree for the goal concept according to the modalities of a concept and optional links. By competing instances, generated for a concept, this search tree is additionally expanded. Based on the judgments for instances, judgments for concepts restricted to one path of the tree can be estimated. This yields to judgments for the nodes of the search tree which can be used for the A^* algorithm. Therefore, the rules for instantiation in connection with the A^* algorithm form the skeleton for different control strategies (see Section IV).

Rules 1–3 are sufficient to define the instantiation of concepts if no modified concepts are allowed. As mentioned above, a modified concept of A can be computed if some instances have been computed, but instantiation of A is not yet possible. For example, having an instance of N in Fig. 9 may allow one to compute a more refined range of attribute values for A. This bottom-up creation of modified concepts is summarized in Rule 4 below. On the other hand, having a modified concept of A, this may in turn be used to restrict attribute values of L in a new modified concept of L. This top-down creation of modified concepts is summarized in Rule 5. The situation is similar if not an instance of N is computed, but a modified concept, a case which is also handled by Rules 4 and 5.

Rule 4:

IF for a *concept A* or a *modified concept* $mod_j(A)$, a new modified concept or a new instance were created for a concept, which is referred to as *part*, or *concrete*, or *context-of* by the concept A

THEN create a new *modified concept* $mod_k(A)$ out of A or $\text{mod}_j(A)$, respectively, as follows:

- construct an empty modified concept of A
- connect this modified concept to all instances referred to by the premise and those which are already referred to by $\text{mod}_j(A)$
- activate the functions like in Rule 1.

By building up modified concepts top down, one can use the inverse functions to constrain a concept due to expectations and due to knowledge gained during the analysis. This is done by Rule 5.

Rule 5:

IF for a *concept A* or for a *modified concept* $mod_j(A)$, a new modified concept or a new partial instance were created for a concept Z, which is referred to as *part-of* or *concrete-of* by the concept A

THEN create a new *modified concept* $mod_k(A)$ out of A or $\text{mod}_j(A)$, respectively, as follows:

- activate the functions in the following sequence:

inverse judgment of link description of Z, inverse computation of attributes of Z, inverse judgments of relations of Z, judgments of links of A, computation of attributes

```
mod_n(TRUCK)

  PART: drivers_cab
    GOAL_NODE        in_j(DRIVERS_CAB)

  ATTRIBUTE: height
    RESTRICTION      [2.9 - 3.5]
    VALUE            ---
  ATTRIBUTE: length
    RESTRICTION      [6.0 - 12.0]
    VALUE            ---

  STRUCTURAL_RELATION: height_smaller_length
    VALUE            1.0 (optimistic)
  ANALYSIS_RELATION: wheels_down
    VALUE            1.0 (optimistic)

  JUDGEMENT: 0.97 (optimistic)
```

Fig. 12. The modified-concept $\text{mod}_n(\text{TRUCK})$.

of A, judgment of attributes of A, judgment of relations of A, judgment of A.

After the instantiation of DRIVERS_CAB, the premise of Rule 4 is satisfied for the concept TRUCK [see Fig. 7(a)]. The restrictions made in the modified concept $\text{mod}_n(\text{TRUCK})$ are illustrated in Fig. 12. Notice that no value, but only a new restriction is calculated for the attribute "height." With $\text{mod}_n(\text{TRUCK})$, the premise for Rule 5 becomes true for the concept WHEEL and the concept CARGO_SPACE. The resulting modified concept $\text{mod}_l(\text{WHEEL})$ of the link "front_wheel" is shown in Fig. 13(b). Since the concept WHEEL may be also a part of a concept CAR, the lower bound of the restriction of the attribute "radius" is 0.20. The minimum radius of a truck wheel, however, is 0.3. This restriction on the links "front_wheel" and "rear_wheel" can be inverted by constraining the lower bound of the attribute "radius" to 0.3. Furthermore, by the application of the inverse calculation of the attribute "height" in the concept TRUCK, the higher bound of the restriction of the attribute "radius" in $\text{mod}_l(\text{WHEEL})$ can be restricted additionally because a radius of more than 0.5 would exceed the maximum value of the restriction of $\text{mod}_n(\text{TRUCK})$. Analogously by the inverse judgment of the relation "wheels_down," the restriction of the attribute "center" can be restricted, as one knows the values of the angles of the instance $\text{in}_j(\text{DRIVERS\_CAB})$.

In order to incorporate results of initial segmentation in a bottom-up manner, a sixth rule is introduced. It uses the attribute and concept lists introduced in Section IV below. Basically, an attribute list contains tuples [concept, role, type] computed from the knowledge base. If from initial segmentation a segmentation object having an attribute with a certain role and type is obtained, this may indicate the occurrence of the concept in the corresponding row of the attribute list. A concept list contains the names of all concepts having a set of tuples [role, type]. One or several attributes are not sufficient in general to compute an instance of the concept, but they may be used to compute a modified concept and its judgment. This is summarized in the following.

Rule 6:

IF initial segmentation provides attributes with [role, type], AND there is a concept K in some row of the attribute list or concept list having the roles and types among its attributes,

THEN create a modified concept $\text{mod}(K)$ of K as follows: generate an empty modified concept, connect it to those referenced in the premise, activate functions in the order of Rule 1.

```
WHEEL

  CONCRETE: geometry
    GOAL_NODE:       CIRCLE

  ANALYSIS_PARAMETER: radius
    TYPE_OF_VAL:     REAL
    NUMB_OF_VAL:     1
    RESTRICTION:     [0.25 - 0.75]
    COMP_OF_VAL:     compute_radius
      ARGUMENT:        geometry.radius
      INV_FUNCT        inv_compute_radius
  ANALYSIS_PARAMETER: centre
    TYPE_OF_VAL:     RECORD, point
    NUMB_OF_VAL:     1
    RESTRICTION:     [ (0,0) - (512,512) ]
    COMP_OF_VAL:     compute_centre
      ARGUMENT:        geometry.centre
      INV_FUNCT        inv_compute_centre

  JUDGEMENT: judge_wheel
    ARGUMENT:        geometry, radius, centre
```

(a)

```
mod_l(WHEEL)

  ANALYSIS_ATTRIBUTE: radius
    RESTRICTION      [0.35 - 0.6]
    VALUE            ---
  ANALYSIS_ATTRIBUTE: centre
    RESTRICTION      [ (30,0) (100,100) ]
    VALUE            ---

  JUDGEMENT: 0.95 (optimistic)
```

(b)

Fig. 13. (a) The concept WHEEL. (b) The modified concept $\text{mod}_l(\text{WHEEL})$.

The above six rules precisely define the computation of instances and modified concepts. As mentioned above, they are the basis for problem-independent control, but they are not sufficient to guarantee efficient analysis of image or speech signals. The reason is the ambiguity of these signals causing the computation of competing instances. An efficient control strategy should focus on the most promising instances and avoid the useless ones. Approaches to such control strategies are treated in the next section.

IV. Problem-Independent Control

In a first realization of the knowledge structure [41], the control strategy can be described as a strict top-down graph search algorithm within the semantic network. In that system, the user interactively selects a goal concept from the network. The system task then is to instantiate the chosen goal concept, that is, to verify the goal concept. In other words, the control algorithm is trying to build up an instance of the concept with respect to the input data. The instantiation of a goal concept can be divided into two steps; the first step is a top-down expansion of the network to find all concepts which are a necessary prerequisite for the instantiation of the goal concept; the second step is a bottom-up instantiation of expanded concepts. The rules for instantiation (Rule 1, Rule 2, and Rule

3) together with the A^* algorithm form a first complete control strategy and build the base for extended strategies. Before describing some extensions, a sketch of the two main steps of the basic strategy will be given. For simplicity, it is assumed in the following that there are no dependencies on the context and no sets of modality, and that any attribute of a concept only depends on attributes of parts directly linked to it.

Given a goal concept, the control starts with top-down expansion of the network. As mentioned in Section III-C, the instantiation rule Rule 1 is applicable to every concept of the network. It determines which concepts are a necessary prerequisite for the instantiation of that concept. Before instantiating the goal concept, first there must be instances of parts and concretes directly linked to that concept. In general, these concepts themselves have parts and concretes, and so on. So one has to expand the network top down along the "part" and the "concrete" links, starting with the goal concept. This expansion process stops at the level of minimal concepts. Minimal concepts are defined by having no parts and no concretes (see Section III-A). It is assumed that minimal concepts can be instantiated immediately on image (or speech) data, using results from image preprocessing and segmentation. After instantiation of such minimal concepts, successive concepts on higher levels can be instantiated, until the goal concept is reached. Network expansion is done in a depth-first manner. Because of ambiguities arising by the possibility of instantiating a concept in different ways on the same input data, there may be more than one instance for a concept. To handle this problem of competing instances, a search algorithm based on the A^* algorithm is used.

This simple presentation of expansion and instantiation should be enough to give an idea of what is meant. For further details, see [41]. It should be noted that the process of expanding a goal concept is simplified by the restrictions defined in Fig. 4. By these restrictions, cycles are avoided which otherwise would have to be tested for during expansion.

This control algorithm has the advantage of being conceptually simple and sufficient for moderately complex problems. But it also has two weak points. First, the selection and instantiation of a goal concept results in a strictly top-down analysis. The processing is model-driven, and there are no possibilities for the control to act flexibly with respect to the input data. Second, the interactive selection of a goal concept by the user may be a critical point. If the network contains a lot of concepts and the user always selects the most general concept, the control algorithm always expands and instantiates a great part of the network. In order to avoid this, the user may select a goal concept at a lower level of the network. This reduces the number of concepts to be expanded and instantiated, but it leaves the problem of goal selection to the user.

Because of this, another more flexible control strategy has been designed, allowing a mixed top-down and bottom-up analysis without interaction of the user. A crucial point of this bidirectional control algorithm is the use of attributes leading to additional knowledge sources that can be used to direct analysis and limit the search space. There are several possibilities for making use of attributes. One version of a bidirectional control algorithm only uses attributes whose set of values is a finite set. Examples are a finite set of colors, shapes, or locations. Details of this approach are given in [37]. A more general approach is described in this paper.

The idea of the bidirectional control algorithm presented here is to determine an attribute list by the rule

IF (a concept has an attribute with a certain role and type),

THEN (insert the triple [concept, role, type] into an attribute list).

Examples of tuples (attribute.role, attribute.type) are (length, real), (color, set of colors), or (speed, real). From initial segmentation, one obtains segmentation objects like lines, regions, or vertices having certain attributes. Using the attributes of segmentation objects and the attribute list, a set A of concepts possibly corresponding to these attributes is determined. The selection of concepts can be constrained more tightly by a concept table. This contains the names of all concepts having a certain set of tuples [role, type], not just one.

So far, the value of an attribute has not been used. The first step of the control algorithm is to compute a modified concept for every concept in the set A using Rule 6. The function to compute a judgment is invoked for every modified concept; the judgment was outlined in Section III-B3). From this discussion, it is evident that the judgment of an attribute influences the judgment of a modified concept. For example, assume a line L from initial segmentation to have (attribute.role = length, attribute. type = real, attribute.value = 10 m), and two concepts X and Y to have (attribute.role = length, attribute. type = real, attribute.restriction = (10 m, 20 m) and (attribute.role = length, attribute.type = real, attribute.restriction = (40 m, 100 m), respectively. Let us assume that the judgment functions of the two attributes of X and Y are computing the degree of membership by using fuzzy membership functions. Irrespective of the precise fuzzy membership functions, the degree of membership of the attribute of L when given X will be high, and that of L when given Y will be low. By this approach, a judgment of every modified concept is computed. The modified concepts are viewed as nodes in a search space. Fig. 14 gives an overview of the complete algorithm. Its idea is to apply the A^* algorithm to the search space initialized by the modified concepts. In every step of the algorithm, the best node in the search space is selected, and it is tried to compute additional instances.

The modified concepts obtained from segmentation results will be on a fairly low level of abstraction. As pointed out in Section III-B2), different levels of abstraction or different conceptional systems are represented by the hierarchy of concrete links in the network. A node in

```
initialize the search space S by the root node R
from segmentation results and concept tables compute a set A of
concepts corresponding to segmentation results
FOR each concept K in set A DO:
    generate one successor node V of the root R in the search space S
    apply RULE6 to K to compute a modified concept mod(K)
    the judgement of V is equal to the judgement of mod(K)
    generate list P(V) for concept K and node V
    Z(V) = K
WHILE specified degrees of concretes and parts have not yet been
      achieved DO:
    select the node N with best judgement in search space S
    IF:    P(N) is not empty
    THEN:  choose the first concept C on P(N)
           instantiate C by using RULE1 and RULE2
           FOR each instance of C DO:
               generate one successor node O of N
               compute judgement of O by rules of algorithm A*
               generate list P(O) for the node O
               remove C from P(N); Z(O) = Z(N)
    ELSE:  determine a set B of superior concepts of Z(N)
           FOR each concept K in set B DO:
               generate one successor node V of N in space S
               apply RULE4 to K to compute a modified concept mod(K)
               the judgement of V is equal to the judgement of mod(K)
               generate list P(V) for K and V, Z(V) = K
```

Fig. 14. An outline of a bidirectional control algorithm. RULE$_i$ refers to the corresponding rule described in Section III-C. The generation of list P is described in Fig. 15.

the search space may represent an uninstantiated concept (in which case the premise list P in Fig. 14 is not empty; this list follows from Rules 1 and 2) or an instantiated concept (in which case P is empty). Since initially concepts are on a low level of abstraction, one will not want to stop on this level. Therefore, a set B of superior concepts is determined where "superior" may be defined in different ways. An approach is to determine less concrete and less general concepts, that is, higher in degrees $d1$ and $d2$. Rule 4 is used to generate modified concepts for the concepts in B. A node in the search space is generated for every modified concept and a judgment is computed.

The algorithm ends if a specified level of abstraction (or a specified degree of the concrete and part links of an instantiated concept) has been reached. It initializes modified concepts in a bottom-up phase. It alternates between top-down instantiation of a node in the search space and bottom-up determination of superior concepts. The activity is directed only by judgments of nodes in the search space according to the strategy of the A^* algorithm. It is more general than the previous algorithm in [37] since it makes use of all attributes, and it is more uniform since it uses only one uniform search space, whereas the former algorithm used several search spaces, the so-called local search spaces.

Although the above algorithm is more complex and more powerful than the strict top-down algorithm outlined in the beginning, it still does not make full use of the potentials of the network. In fact, it only uses Rules 1, 2, 4, and 6 in Section III-C. So it is possible to design other control algorithms. However, we think the most important point is that all those algorithms will be special versions of graph search algorithms. The combination of semantic networks and graph search thus provides a framework to treat the control problem and to design different classes of control algorithms which can be used in various task domains.

V. Network Environment

In order to provide a comfortable environment for working with semantic networks, it is necessary to have tools which support the generation, the modification, and the consistency test of a semantic network, and an easy utilization of the stored knowledge. Those tools which are written in C are implemented in the network environment of ERNEST, including an editor for easy manipulation of networks. Furthermore, access routines to the network structures and routines for developing control algorithms are available.

Besides the tools for creating and manipulating a semantic network, there are programs which allow a simple utilization of the coded knowledge for an analysis process. The following tools are feasible if a network for a special application was successfully tested for its consistency. To make knowledge utilization more efficient, the network can be prepared for an analysis process. This means that all information necessary for a control algorithm is computed in advance whenever possible. For all concepts, the substructures are calculated which are inherited from concepts referred to "specialization of" links. Pointers to these are inserted in the considered concept. The gain in efficiency is obvious because for the access of a substructure, no search along the links "specialization of" is necessary. Since for the calculation of attributes and analysis parameters values of attributes and analysis parameters of the same concepts also can be used, it is necessary that these self-references are consistent. In addition, a certain order for the calculation has to be followed, and therefore, the so-called *concept flowchart* is created. On the one hand, it indicates the order of calculation, and on the other hand, it represents the self-references between the arguments.

A further tool in the preparing of a network for the analysis is the calculation of the *initial search graph* for every concept. It represents all possible paths for the instantiation of this concept which result from the first two rules for instantiation; this corresponds to the list P in Fig. 15. This information also increases the efficiency of an analysis because the search tree for an analysis process is closely connected to the initial search graph.

In a network prepared in this way, one can use tools for building up instances and modified concepts. According to the pragmatics of the network (see Section III-C), there are routines which create partial instances, which complete partial instances, and which extend instances by optional links. Furthermore, one can create modified concepts and modify modified concepts top down and bottom

```
compute_list_(K,P)
FOR all concepts A [(which are a context independent part of K
    OR which are a concrete of K) AND (where an instance of A is
    not yet on node V(K)] DO:
    compute_list(A,P)
IF   (K has context dependent parts),
THEN (attach K_partial to the end of P)
FOR all concepts A which are a context dependent part of K
    AND where an instance of A is not yet on node V(K) DO:
    compute_list(A,P)
attach K to the end of P
```

Fig. 15. Algorithm for the computation of list P for concept K and search space node $V(K)$ corresponding to K.

up. These complex routines are powerful tools for the analysis because they execute the main steps of analysis independently of a concrete control algorithm, namely, local adaptation of the knowledge base and instantiation of concepts and modified concepts.

VI. Applications

At our Institute, three pattern analysis applications use the ERNEST system. The problems attacked by these applications are quite different. Common to all is that sensor data input is used. The differences between the tasks and the underlying data indicate the epistemological adequacy of the network approach and also the problem-independent structure of the total ERNEST system.

A. Scintigraphic Image Analysis

The task of this system is to automatically give a description in diagnostic terms of image sequences of the heart. The input data are Tc–99m gated blood pool studies with a spatial resolution of 64×64 pixels and 12–32 images per sequence. A sequence represents the motility behavior of the heart between two R waves of the ECG. An example of such an image sequence filtered by a 7×7 median is shown in Fig. 16. The knowledge base of the system consists of about 180 concepts. A condensed view of the network is shown in Fig. 17. Each block in this figure stands for a collection of concepts having identical degree with respect to the specialization and concrete links. Inside a block, concepts are connected via part links. The network consists of eight conceptional systems from the interface system at the bottom to the complete interpretation system at the top. In between the medical objects, the frame-to-frame motility, motility paths, segmentation of the left ventricle volume curve, diagnostic interpretations for the motility of the different objects, and the interpretations of the form and proportions of the heart are represented. In the total network, a few hundred links, attributes, and relations are defined. An analysis process is monitored by the A^* algorithm. For the judgments of instances [in the sense of Section III-B3)] and search graph nodes, fuzzy logic is used as certainty measurement.

A detailed description of this system and the results are reported in [31] and [41]. Therefore, we only mention

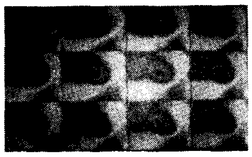


Fig. 16. An image sequence of the motility behavior of the heart.

that in a test of 21 image sequences, 18 were interpreted correctly and three were rejected.

B. Analysis of Industrial Scenes

The purpose of analysis of industrial scenes may be the detection of objects, fault detection, or working area control. Therefore, the description as the result of analysis differs depending on the purpose. In any case, the analysis starts with processing one image or one stereo-image pair. Image processing results are edges and regions with associated features. These features may be 2-D, 2.5-D, or 3-D features.

The purpose of the image analysis system that we are presently implementing is workpiece recognition and detection for flexible automatic manufacturing. The algorithms used for feature extraction include stereo and shape-from-shading algorithms. Using a model represented by a semantic network, a combination of the above-described control algorithms monitors the analysis.

To make the image analysis system flexible enough as is necessary for modern manufacturing, we have integrated a knowledge acquisition component in ERNEST. This component itself can be easily adopted to a special task domain. A description of it is omitted here due to space limitations. Details can be found in [43]. The input of the knowledge acquisition consists of images of the object that has to be learned and of the corresponding CAD data.

A preliminary version of the image analysis system has been completed. It uses a contour-based segmentation and models of the views of stable positions of the workpieces. The assembly of electric motors is an experimental task where this system has been used for object identification and location determination. Different tests of the image analysis system have shown that it reliably recognizes and localizes parts under variable viewing conditions. The automatically generated models consist of 5–100 concepts, depending on the complexity of the object. The analysis time on a VAX Station 3200 took from 2 CPU s up to 5 CPU min. By not using all features of an object but only the most robust ones, the time can be reduced to less than 10 CPU s for all objects. The location defect was about two pixels. The results have shown that the requirements of an industrial environment can be met by an image analysis system that has been developed under ERNEST.

C. Speech Understanding

The aim of the project EVAR [30], [36] is the automatic understanding of continuous German speech. Therefore, a homogeneous hierarchical knowledge base

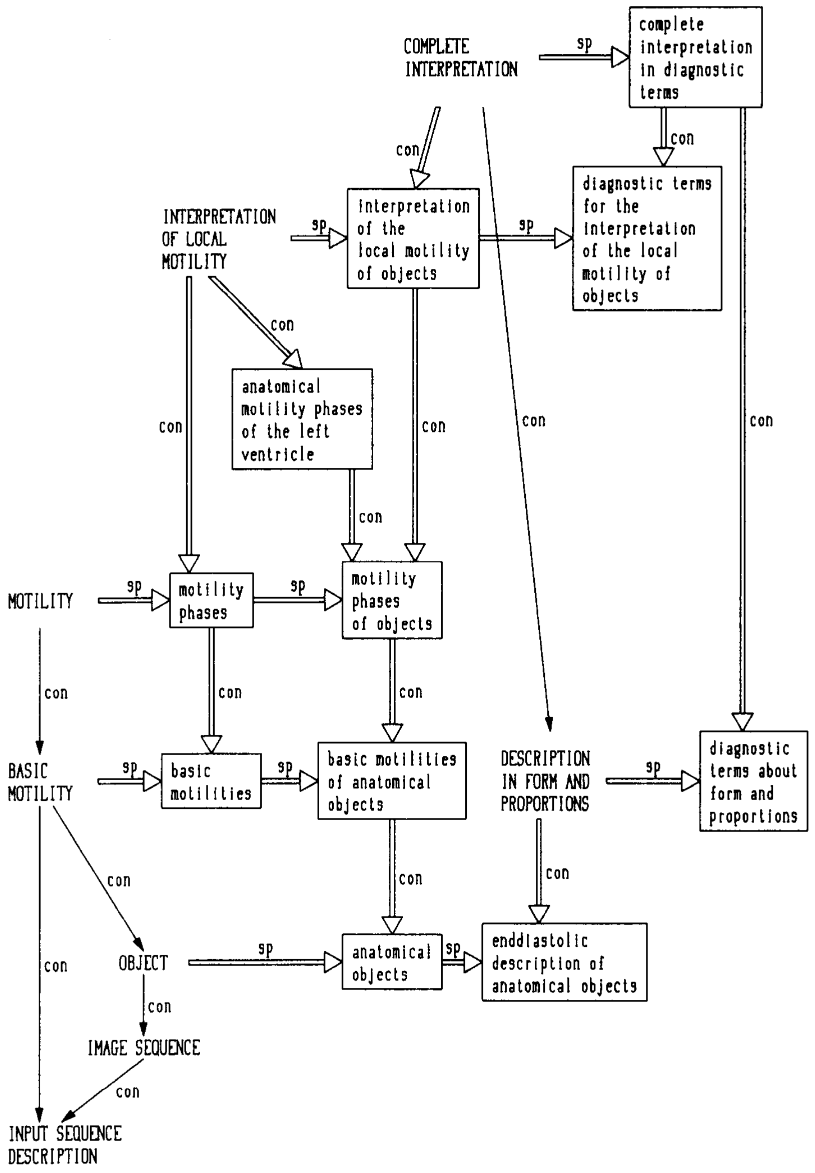

COMPLETE INTERPRETATION
sp
complete interpretation in diagnostic terms
con
diagnostic terms for the interpretation of the local motility of objects
INTERPRETATION OF LOCAL MOTILITY
interpretation of the local motility of objects
anatomical motility phases of the left ventricle
MOTILITY
motility phases
motility phases of objects
BASIC MOTILITY
basic motilities
basic motilities of anatomical objects
DESCRIPTION IN FORM AND PROPORTIONS
diagnostic terms about form and proportions
OBJECT
anatomical objects
enddiastolic description of anatomical objects
IMAGE SEQUENCE
INPUT SEQUENCE DESCRIPTION

Fig. 17. Overview of the network for diagnostic description of scintigraphic image sequences.

[34] was created based on existing work. It represents the syntax and semantics of the German language and the prototypical task domain "intercity-train-information." Fig. 18 gives a detail of the knowledge base.

In the lowest level, there are concepts (i.e., H_ WORD_HYPOTHESIS) which build the main interface between word recognition and linguistic analysis. The instances of those concepts are computed from the actual set of word hypotheses. Besides that, on other locations in the network, communication takes place between these two parts of analysis. So one can create hypotheses based on modified concepts or instances which also use linguistic knowledge for the judgment, or one may verify a word chain.

The basis for the syntactic judgment of an interpretation is the concept SYNTAX and its specialization. The syntactic classes (i.e., SY_NOUN, SY_PRON) are modeled here, which are the connection to the lowest level. Based on those, greater syntactical units, i.e., noun phrase (SY_NP), are described by the links *part*.

The next level contains concepts which represent the semantics based on the case grammar [9]. The concepts for deep cases opened by noun frames, verb frames, and adjective frames are connected to the concepts representing these frames.

The highest level reflects the pragmatics given by the task domain "intercity-train-information." Three kinds of concepts are distinguished, namely, goal concepts

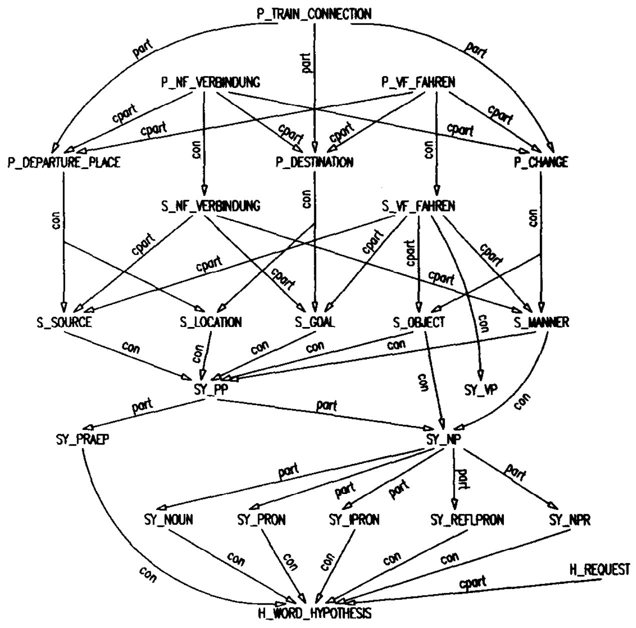


Fig. 18. Detail of a network for speech understanding covering the abstraction levels of words, syntax, semantics, and pragmatics.

(i.e., P_TRAIN_CONNECTION), frames of meaning (i.e., P_V_FAHREN), and pragmatical intentions (i.e., P_DESTINATION). The last two kinds of concepts are analogous to the frames and deep cases in the semantics.

The first tests were done with a knowledge base of about 120 concepts and with a lexicon of 1600 items. For the analysis, the 100 best scored word hypotheses were used and augmented by the deficient correct word hypothesis without changing the score. In an initial phase, the ten best scored word hypotheses with a relevant pragmatical class were used to estimate possible goal concepts which the control algorithm, described in Section IV, tried to instantiate. For two sentences, the analysis could be finished successfully. This means that the correct pragmatical instance (P_TIME_TABLE_INFO) with the correct pragmatical intentions was built up. For four other sentences, the analysis did not come to an end due to storage limitation (more than 1500 search tree nodes). The reason was a very bad rank (less than 2000) of some correct word hypotheses on the boundaries of the speech signal because these regions were insufficiently digitized. For the two recognized sentences, 640/760 search tree nodes were expanded, 600/715 instances, and 703/824 modified concepts were created. On a VAX Station 2000, the analysis consumed about 12 CPU min and about 8 Mbyte storage. As one can see, the number of instances and modified concepts increases approximately linearly with the number of search tree nodes.

VII. Conclusions and Outlook

A system environment based on a special definition of a semantic network has been described which is specially designed for knowledge-based image and speech understanding. It allows one to represent high-level task-specific knowledge on different levels of abstraction. Low-level (or task-independent) process is incorporated into the system via minimal concepts and their direct instantiation. The use of the knowledge base is defined by six task-independent rules for the instantiation and modification of concepts. The rules allow one to design different control algorithms for the analysis of an image or an utterance. Appropriate tools for the design, test, and modification of knowledge bases are available, and the syntax of the network is checked automatically. The approach is being used successfully in three different application areas, two in image understanding, and one in speech understanding.

We are presently extending the approach in the direction of providing a separate explanation facility which is designed to work on the network structure, but which should be task-independent to a large extent. Another plan for the future is to explore and design facilities for a truth-maintenance system.

It has been demonstrated in [44] that computations in a particular type of semantic network can be done in a time order depending on the depth of the network—in our terminology, depending on the order of the concretization hierarchy. Another direction for future research thus may be to explore parallel instantiation of concepts in our more elaborate network structure.

References

[1] D. H. Ballard and C. M. Brown, *Computer Vision.* Englewood Cliffs, NJ: 1982.

[2] R. J. Brachman, "On the epistemological status of semantic networks," in N. V. Findler, Ed., *Associative Networks*. New York: Academic, 1979.

[3] R. J. Brachman and I. A. Schmolze, "An overview of the KL-ONE knowledge representation language," *Cognitive Sci.*, vol. 9, pp. 171-216, 1985.

[4] R. Brooks, "Symbolic reasoning among 3-D models and 2-D images," *Artificial Intell.*, vol. 17, pp. 285-341, 1981.

[5] R. De Mori, *Computer Models of Speech Using Fuzzy Algorithms*. New York: Plenum, 1983.

[6] W. Eichhorn and H. Niemann, "A bidirectional control strategy in a hierarchical knowledge structure," in *Proc. 8th Int. Conf. Pattern Recognition*, Paris, France, 1986, pp. 181-183.

[7] W. Eichhorn, "Eine aktive hierarchische Wissensstruktur für die Musteranalyse," dissertation, Lehrstuhl für Informatik 5 (Mustererkennung), Universität Erlangen-Nürnberg, Erlangen, W. Germany, 1988.

[8] L. D. Erman and V. R. Lesser, "The Hearsay-II speech understanding system: A tutorial," in W. A. Lea, *Trends in Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1980, pp. 361-381.

[9] C. J. Fillmore, in *The Case for Case. Universals in Linguistic Theory*, E. Bach and R. T. Harms, Eds. New York: Holt, Rinehart, and Winston, pp. 1-88.

[10] N. V. Findler, Ed., *Associative Networks*. New York: Academic, 1979.

[11] A. R. Hanson and E. M. Riseman, Eds., *Computer Vision Systems*. New York: Academic, 1978.

[12] J. P. Hayes, "The logic of frames," in D. Metzing, Ed., *Frame Conceptions and Text Understanding*. Berlin: de Gruyter, 1979, pp. 46-61.

[13] F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, Eds., *Building Expert Systems*. New York: McGraw-Hill, 1982.

[14] C. Hewitt, "PLANNER: A language for proving theorems in robots," in *Proc. IJCAI 2*, 1971.

[15] —, "Viewing control structures as patterns of passing messages," *Artificial Intell.*, vol. 8, pp. 323-364, 1977.

[16] I. Hofmann, R. Gämlich, and H. Niemann, "A human interface for control of an image processing system," in *Proc. 8th Int. Conf. Pattern Recognition*, Paris, France, 1986, pp. 1256-1258.

[17] R. Kowalski, "Predicate logic as a programming language," in *Information Processing 74*. Amsterdam: North-Holland, 1974, pp. 569-574.

[18] F. Kummert, H. Niemann, G. Sagerer, and S. Schröder, "Werkzeuge zur modellgesteuerten Bildanalyse und Wissensakqusition—Das System ERNEST," in M. Paul (Hrsg.), *GI-17. Jahrestagung Computer-integrierter Arbeitsplatz im Büro*. Berlin: Springer-Verlag, 1987, pp. 556-570.

[19] W. A. Lea, *Trends in Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[20] H. Levesque and J. Mylopoulos, "A procedural semantics for semantic networks," in N. V. Findler, Ed., *Associative Networks*. New York: Academic, 1979, pp. 93-121.

[21] Z. Manna, *Mathematical Theory of Computation*. New York: McGraw-Hill, 1974.

[22] J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in B. Meltzer and D. Ritchie, Eds., *Machine Intelligence 4*. Edinburgh, Scotland: Edinburgh Univ. Press, 1969, pp. 463-502.

[23] D. McDermott, "The last survey of representation of knowledge," in *Proc. AISB Conf.*, Hamburg, W. Germany, 1978, pp. 206-221.

[24] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., *Machine Learning, An Artificial Intelligence Approach*. Palo Alto, CA: Tioga, 1983.

[25] M. Minsky, "A framework for representing knowledge," in P. H. Winston, Ed., *The Psychology of Computer Vision*. New York: McGraw-Hill, 1975, pp. 211-277.

[26] J. Mylopoulos and H. Levesque, "An overview of knowledge representation," in M. Brodie, J. Mylopoulos, and J. V. Schmidt, Eds., *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. New York: Springer-Verlag, 1983.

[27] M. Nagao and T. Matsuyama, *A Structural Analysis of Complex Aerial Photographs*. New York: Plenum, 1980.

[28] A. M. Nazif and M. D. Levine, "Low level image segmentation: An expert system," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 555-577, 1984.

[29] H. Niemann, *Pattern Analysis*. Berlin: Springer, 1981.

[30] H. Niemann and G. Sagerer, "Semantische Netze als Ansatz zur Repräsentation von Wissen für die automatische Bildanalyse," *Robotersysteme 1*, pp. 139-150, 1985.

[31] H. Niemann, H. Bunke, I. Hofmann, G. Sagerer, F. Wolf, and H. Feistel, "A knowledge based system for analysis of gated blood pool studies," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 246-259, 1985.

[32] H. Niemann, "A homogeneous architecture for knowledge based image understanding," in *Proc. 2nd Conf. Artificial Intell. Appl.*, Miami, FL, 1985, pp. 88-93.

[33] H. Niemann, A. Brietzmann, R. Mühlfeld, P. Regel, and G. Schukat, "The speech understanding and dialog system EVAR," in R. De Mori and S. Y. Suen, Ed., *New Systems and Architectures for Automatic Speech Recognition and Synthesis*, NATO ASI Series F16. Berlin: Springer, 1985, pp. 271-302.

[34] H. Niemann, A. Brietzmann, U. Erhlich, and G. Sagerer, "Representation of a continuous speech understanding and dialog system in a homogeneous semantic net architecture," in *Proc. ICASSP'86*, Tokyo, Japan, 1986, pp. 1581-1584.

[35] H. Niemann and H. Bunke, *Künstliche Intelligenz in Bild- und Sprachanalyse*. Stuttgart: Teubner, 1987.

[36] H. Niemann, A. Brietzmann, U. Ehrlich, S. Posch, P. Regel, G. Sagerer, R. Salzbrunn, and G. Schukat-Talamazzini, "A knowledge based speech understanding system," *Int. J. Pattern Recognition Art. Intell.*, vol. 2, pp. 321-350, 1988.

[37] H. Niemann, G. Sagerer, and W. Eichhorn, "Control strategies in a hierarchical knowledge structure," *Int. J. Pattern Recognition Art. Intell.*, vol. 3, pp. 557-572, 1988.

[38] N. J. Nilsson, *Principles of Artificial Intelligence*. Berlin: Springer, 1982.

[39] W. A. Perkins, "INSPECTOR: A computer vision system that learns to inspect parts," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 584-592, 1986.

[40] H. Prade, "A computational approach to approximate and plausible reasoning with applications to expert systems," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 260-283, 1985.

[41] G. Sagerer, "Darstellung und Nutzung von Expertenwissen für ein Bildanalysesystem," in *Informatik Fachberichte*. Berlin: Springer, 1985.

[42] P. Schefe, *Künstliche Intelligenz—Überblick und Grundlagen*. Mannheim, Wien, Zürich: Bibliographisches Institut, 1986.

[43] S. Schröder, H. Niemann, and G. Sagerer, "Knowledge acquisition for a knowledge based image analysis system," in *Proc. European Knowledge Acquisition Workshop (EKAW88)*, Gesellschaft für Mathematik und Datenverarbeitung mBH, Bonn, W. Germany, 1988, pp. 29/1-29/15.

[44] L. Shastri, *Semantic Networks: An Evidential Formalization and Its Connectionist Realization*. London: Pitman, 1988.

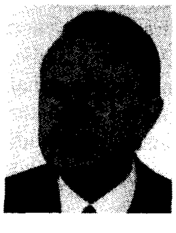
[45] Y. Shirai, *Three-Dimensional Computer Vision*. Berlin: Springer, 1987.

[46] J. K. Tsotsos, "A framework for visual motion understanding, "Ph.D. dissertation, Dep. Comput. Sci., Univ. Toronto, Toronto, Ont., Canada, 1980.

[47] P. H. Winston, *Artificial Intelligence*, 2nd ed. Reading, MA: Addison-Wesley, 1984.

[48] W. Woods, "What's in a link. Foundations for semantic networks," in D. Bobrow and A. Collins, Ed., *Representation and Understanding*. New York: Academic, 1975.

[49] M. Yashida and S. Tsuji, "A versatile machine vision system for complex industrial parts," *IEEE Trans. Comput.*, vol. C-26, pp. 882-894, 1977.

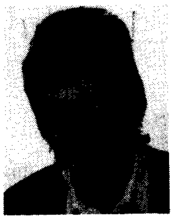


Heinrich Niemann (M'77) received the Dipl.-Ing. degree in electrical engineering and the Dr.-Ing. degree from the Technical University of Hannover, Hannover, West Germany, in 1966 and 1969, respectively.

From 1967 to 1972 he was with Fraunhofer Institut für Informationsverarbeitung in Technik ung Biologie, Karlsruhe, working in the field of pattern recognition and biological cybernetics. During 1973-1975 he was teaching in the Department of Electrical Engineering, Fachhochschule Giessen. Since 1975 he has been a Professor of Computer Science at the University of Erlangen-Nürnberg, Erlangen, West Germany, and since 1988 he has also been Head of the research group "Knowledge Processing" at the Bavarian Research Institute for Knowledge Based Systems (FOR-

WISS). His fields of research are image and speech understanding and the application of artificial intelligence techniques in these fields. During 1979–1981 he was Dean of the Engineering Faculty of the University, in 1982 he was Program Chairman of the 6th International Conference on Pattern Recognition, München, and in 1987 he was Director of the NATO Advanced Study Institute on "Recent Advances in Speech Understanding and Dialog Systems." He is on the Editorial Board of *Signal Processing* and *Pattern Recognition Letters* and is the author, coauthor, or editor of nine books and more than 100 technical papers.

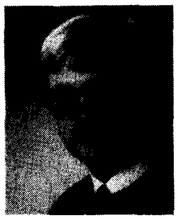
Dr. Niemann is a member of EURASIP, GI, and VDE.



Gerhard F. Sagerer (M'88) received the diploma and the Ph.D.(Dr.-Ing.) degree in computer science from the University of Erlangen-Nürnberg, Erlangen, West Germany, in 1980 and 1985, respectively.

His research interest is knowledge representation and control algorithms and their application in the field of image and speech understanding. From 1980 to 1985 he worked on the automatic interpretation of scintigraphic image sequences of the human heart. Since 1984 he has been working on projects for knowledge-based speech understanding and the interpretation of industrial scenes. He has published numerous papers in these fields, is author of a book on knowledge representations and its use in the application of image analysis, and coeditor of a book on speech understanding and dialog systems.

Dr. Sagerer is a member of the German Computer Society (GI) and the European Society for Signal Processing (EURASIP).



Stefan Schröder received the Dipl.-Inf. degree in informatics (computer science) from the University of Erlangen-Nürnberg, Erlangen, West Germany, in 1985.

Since 1985 he has been with the Institut für Informatik, Mustererkennung (pattern recognition), University of Erlangen-Nürnberg. His field of research is knowledge-based image analysis, knowledge acquisition, and artificial intelligence.

Franz Kummert was born in Hahnbach, West Germany, on November 18, 1960. He received the Dipl.-Inf. degree in informatics (computer science) in 1986 from the University of Erlangen-Nürnberg, Erlangen, West Germany.

Since 1987 he has been with the Institut für Informatik, Mustererkennung (pattern recognition) at the University of Erlangen-Nürnberg. His field of research is knowledge-based speech understanding.