

AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs

H. Penny Nii and Nelleke Aiello
Heuristic Programming Project, Computer Science Department
Stanford University, Stanford, Ca. 94305

The goal of the AGE project is to demystify and make explicit the *art of knowledge engineering*. It is an attempt to formulate the knowledge that knowledge engineers use in constructing knowledge-based programs and put it at the disposal of others in the form of a software laboratory. To achieve this goal, the task for AGE is divided into two main sub-tasks: (1) isolating techniques used in knowledge-based systems and programming those that are general and useful (2) building an intelligent agent to guide in the use of these techniques. Currently AGE has a facility to build programs using the Blackboard Model [8,13].

1. INTRODUCTION

This paper reports the goals and the current status of the AGE project.^{*} Appendix I contains a protocol of a user solving a specific problem using AGE. The intent of the protocol is to show AGE from the user's point of view. The main body describes the motivations for and the description of the facilities in AGE.

1.1 Objectives

The general goal of the AGE project is to demystify and make explicit the art of knowledge engineering. It is an attempt to formulate the knowledge that knowledge engineers use in constructing knowledge-based programs and put it at the disposal of others in the form of a software laboratory.

The design and implementation of the AGE program is based primarily on the experience gained in building knowledge-based programs at the Stanford Heuristic Programming Project in the last decade. The programs that have been, or are being, built are: DENDRAL, meta-DENDRAL, MYCIN, HASP, AM, MOLGEN, CRYSLIS [9], and SACON [2]. Initially, the AGE program will embody artificial intelligence (AI) methods used in these programs. However, the long-range aspiration is to integrate methods and techniques developed at other AI laboratories. The final product is to be a *collection of building-block programs* combined with an *intelligent front-end* that will assist the user in constructing knowledge-based programs. It is hoped that AGE will speed up the process of building knowledge-based programs and facilitate the dissemination of AI techniques by: (1) packaging common AI software tools so that they need not be reprogrammed for every problem; and (2) helping people who are not knowledge engineering specialists write knowledge-based programs.

The task of building such a software laboratory for knowledge engineers is divided into two main sub-tasks:

1. *The isolation of techniques used in knowledge-based systems.* It has always been difficult to determine if a particular problem solving method used in a knowledge-based program is "special" to a particular domain or whether it generalizes easily to other domains. In existing knowledge-based programs, the domain specific knowledge and the manipulation of such knowledge

using AI techniques are often so closely coupled that it is difficult to make use of the programs for other domains.

2. *Guiding the user in the initial application of these techniques.* Once the various techniques are isolated and programmed for use, an intelligent agent is needed to guide the user in the application of these techniques.

1.2. User Profile

The design of the AGE system would depend, to a great extent, on the type of users we expect will benefit most from using AGE. *Initially, AGE is designed for AI scientists familiar with current problem solving techniques, and who can program in the INTERLISP language [19] (since AGE is implemented in INTERLISP); and are familiar with production-rule representations of knowledge [5].* In other words, AGE is initially aimed at people who could conceivably write knowledge-based programs themselves.

For the person in this category, the advantages of using AGE are twofold:

1. The basic system components are already programmed (e.g., rule interpreters and other control mechanisms, traces, explanation modules, and other components basic to many systems).
2. AGE allows the user to experiment with different problem solving techniques without extensive reprogramming.

Eventually, AGE will be able to help the less knowledgeable or less experienced person.

1.3. Outline of the Paper

The term "knowledge engineering" is being used more frequently to refer to the process of writing application programs using primarily AI methods. The historical context in which knowledge engineering arose and the nature of the work are described by Feigenbaum [9]. Section 2 contains a brief description of the task for the AGE program within the context of knowledge engineers' work. It is followed by an overview of the facilities in the current version. Section 4 contains a short description of an organizational method by which frameworks are decomposed to form a basis for a software laboratory. It is followed by a summary section.

2. TASK FOR THE AGE SYSTEM

Currently there are several projects that aim to provide prepackaged tools for knowledge engineers (UNITS [18],

^{*} This research was supported in parts by the Defense Advanced Research Projects Agency under ARPA Contract No. MDA 903-77-C-0322 and the National Institutes of Health Grant No. RR-00785.

EMYCIN [2]) or to provide programming languages (KRL [4], AIMDS [17]). In preprogrammed packages, all, or most, of the paradigm, system design, and implementation choices have been made by the package designers. In languages none, or very few, decisions have been made for the user. Neither approach guides the user in the design and the construction of knowledge-based programs. AGE has been conceived to bridge the gap between these two extremes—to provide some guidance in the *what*, *why*, and *how* of programming; plus some preprogrammed problem-solving frameworks.

Ideally then, AGE is an attempt to define and cumulate knowledge-engineering tools, with rules to guide in the use of these tools. It must itself be a knowledge-based system containing knowledge about building knowledge-based programs, combined with a facility that allow the user to explore and experiment with various concepts and techniques. It must provide the user with a variety of preprogrammed modules, allow her to modify them and to add her own. And above all, it must be able to produce running programs. How the AGE system itself is organized for modularity will be described in Section 4. In the next section we describe a framework currently available to the user to construct her knowledge-based program.

3. PROFILE OF THE CURRENT AGE SYSTEM

To correspond to the two general research goals described in the Introduction, the AGE program is being developed along two separate fronts. The first of these fronts is the development of tools to help the user build a variety of knowledge-based programs—the *generality* front. The second front is the development of *intelligence* in the interaction between the user and the AGE program; i.e., moving from dialogues on *how to use the tools* in AGE to *what tools to use*—the *how-to-what* spectrum described by Feigenbaum [9].

3.1. Currently Implemented Tools

The building-block components currently available to the users have been carefully selected and modularly programmed to be useable in combinations. The current AGE system aims to provide the user with a framework useful for incremental hypothesis formation, known as the *Blackboard Model* [8, 13]. It can be described as follows:

1. There is a global data base (the blackboard) that is used as a means of communication and interaction among the KSs;
2. There are diverse knowledge sources (KSs) that are kept separate and independent; and
3. The KSs respond to changes in the blackboard.

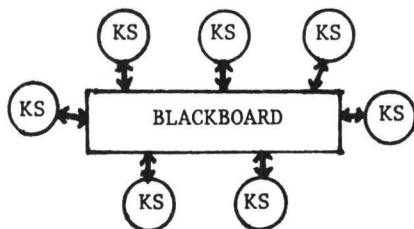


Figure 1. The Blackboard Model

The paradigm itself does not specify the structure of the data base, the representational form of the KSs, nor the response mechanisms. When a model is translated into a program, various architectural and implementation decisions are made by

the designers, and programs that use the same paradigm may have different organization and behavior. For example, although HEARSAY-II [13] and CRYSLIS [7] use the Blackboard model, the basic architecture, knowledge representation, and knowledge utilization techniques differ. The differences can be attributed to many factors: the nature of the problem (understanding speech signals and interpreting x-ray crystallographic data); implementation language; real-time constraints; noise level of data; quality and amount of available knowledge; and, last but not least, the designers' tastes.

In AGE the Blackboard-based program design has been implemented to allow flexibility in representation and in the application of other problem solving methods within the framework. It consists of three major components:

1. **The Blackboard:** The blackboard contains hypotheses in a hierarchical data structure; it represents the task domain in terms of a hierarchy of analysis levels of the task.
2. **The KSs:** The KSs contain the knowledge of the task domain (which the user must provide) that can perform the analysis. The KSs are represented as sets of production rules.
3. **The Control:** The control component contains mechanisms that allow the user to (a) specify the conditions for the invocation of the KSs and (b) to select items on the blackboard for focus of attention [10].

In the remainder of this section these components will be described in more detail.

3.1.1. Blackboard

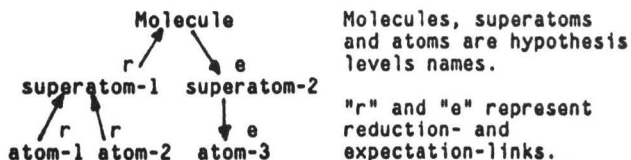
The combined process of KS selection and incremental changes to the blackboard is viewed as a general process of hypothesis formation consistent with the following definition:

An hypothesis can be generated [11]:

1. deductively, using *support from above*—this support can be
 - a. *theoretical* support (or model-based support),
 - b. support from more inclusive hypotheses that have independent evidential support; or
2. inductively, using *evidential support from below*.

This definition suggests that the most natural way to represent hypotheses is in some form of a hierarchy. The structure of the hierarchy may be strict; it may be flat consisting possibly of only two levels—input data and its translation; or, it may be complex, consisting of many related hierarchies (often referred to as blackboard planes [7]). Currently AGE can represent all of the above forms of hierarchy. An hypothesis is represented as hierarchically organized hypothesis elements integrated by links that represent support from above—called the *expectation-link*, or support from below—called the *reduction-link* (see example 1).

EXAMPLE 1: Hypothesis structure in CRYSLIS [7]



An *hypothesis element* is a named node in the hypothesis structure that represents an aggregation (summary,

interpretation, etc.) of lower level hypothesis elements. Each element contains information in the form of attribute-value pairs that are meaningful at that particular hypothesis level (see example 2). These information are inferences generated by the rules in the KSs and can, in turn, be used for further hypothesis formation.

EXAMPLE 2: A possible hypothesis element in CRYSLIS

```
Hypothesis level: ATOMS
Hypothesis element name: ATOMS-18
Inferred attribute-value pairs:
  Atom.name      Sulfur
  Location       (12.3 13.6 24.2 +- .02)
  Partof         (OR CYS14 CYS17)
  Bonded to      HEME-1
  Bondtype       Hydrogen
```

In some problems all the hypothesis elements needed to solve the problems are known *a priori*. In other problems the number of hypothesis elements in the solutions are not known in advance. For example, in PUFF (a program to diagnose pulmonary function disorder [12]) all the possible disease states which account for specific patient data are known in advance. On the other hand, in cryptogram problem (see Appendix I) the number of letters, words, parts of sentence, etc. are not known until a specific cryptogram is being solved. To accommodate both types of solution space, AGE allows the user to generate and name hypothesis elements in advance, generate and name them dynamically, or generate them in some combination.

The hypotheses are generated by inference rules in the KSs. What the KSs look like and how they work are described next.

3.1.2 Knowledge Representation

Knowledge representation in AGE is based on the premise that there are at least four broad categories of knowledge that are needed to solve problems:

1. knowledge of specifics,
2. knowledge of ways and means of dealing with the specifics,
3. knowledge of universals and abstractions in the task domain [3], and
4. knowledge of problem solving and knowledge utilization methods in the task domain.

Within the framework of incremental hypothesis formation using the Blackboard model, an attempt has been made to help the user identify, represent, and utilize these diverse types of knowledge.

Domain knowledge in AGE represents the knowledge necessary to accomplish the goals of the user's program. This knowledge reflects the *knowledge of specifics* and the *knowledge of ways and means of dealing with the specifics*. The knowledge is represented as production rules [5]. These rules are organized into sets called the Knowledge Sources (KSs).

Knowledge Sources

A *Knowledge Source* is a *mega-chunk of knowledge* consisting of a labeled set of rules that are *a priori* deemed to belong together. How the rules within a KS is organized and what rules are included in a KS depend on the intended role of the KS in the overall problem solving plan. For example, a KS may be organized to represent models--in which case all rules are grouped around some objects or concepts, much like schemata or frames. Or, a KS may be organized around

events--in which case all rules which, for example, process input data, are grouped together. In order to solve problems within the framework of the Blackboard model, KSs must place their inferences on the Blackboard. In other words, KSs must generate hypotheses using rules that:

1. add or modify hypothesis element(s), or
2. add or modify relationships between elements by
 - a. analyzing data (support from below),
 - b. specializing or instantiating a more inclusive hypothesis element (support from above), or
 - c. generating expectation(s) from
 1. models, or
 2. more inclusive hypothesis elements that must be verified by data.

Each KS has associated with it: (a) preconditions (a list of events) for its invocation; (b) a list of dotted pairs of hypothesis levels that it spans; (c) list of links generated by it; (d) "single" or "multiple" hit strategy to be used for the rules, and (e) a facility for variable bindings to set local context, to simplify expressions in the rules, to avoid multiple evaluations of the same expression, and to allow communication between the condition and the conclusion halves of a rule.

AGE allows the use of just one KS in the user's program if desired. Whether a single KS is sufficient to solve a problem or not, depends on the characteristics of the problem and on the formulation of the KS. MYCIN [16] is an example of a program that has an equivalent of one KS; all the rules are chained in a goal-directed fashion to represent a line of reasoning meaningful in that domain.

The set of domain-specific KSs may be manipulated in many ways by other higher level KSs [6, 14]. The higher level knowledge may reflect both the knowledge about *ways and means of dealing with the specifics*, *knowledge of problem-solving methods useful in the task domain*, and *knowledge about other knowledge* in the domain. These higher-level KSs are often integrated into the Control component described later.

Rules

A *rule* in AGE is a *chunk of knowledge* written in a syntax comprehensible to AGE. Each rule consists of a left-hand-side (LHS) and a right-hand-side (RHS). The LHS specifies a set of conditions or patterns for the applicability of the rule (i.e. premises). The RHS represents the implications or conclusions to be drawn under the situation specified in the LHS.

To be consistent with the various ways in which hypothesis elements can be generated, the current implementation of the RHS can:

1. add elements to the hypothesis; or add or modify the values of the attributes of elements or the relationship between elements;
2. generate expected elements or values of (or relationship between) elements in the hypothesis;
3. generate goals to establish some elements, or values or links, within hypothesis elements.

Rule Credibility

The domain rules that generate hypotheses are, more often than not, judgmental and uncertain. A *credibility value*, or *weight*, can be associated with the rules to reflect uncertainty. AGE provides means of associating weights (certainty values,

probability, etc.) with inferences generated by the RHS of rules. It also provides some pre-programmed procedures for the manipulation of these weights, for example, an algorithm to compute Certainty Factors (CF) from MYCIN. For users who need different algorithms, AGE provides mechanisms for integrating user-provided algorithms.

EXAMPLE 3: Integrating weights

Note: The rules used in this example, and some others, are based on PUFF rules [12]. From pulmonary function test data, the PUFF program determines if a patient is normal, has obstructive airway disease (OAD), restrictive lung disease (RLD), or other pulmonary function-related diseases; and the severity of the diseases. It further determines if OAD is of type emphysema, asthma, or bronchitis. ("RV" in the example refers to a measurement--residual volume.)

Weight-adjuster: \$ADJWT (MYCIN algorithm)

Rule to be evaluated:

```
if (TYPE = OAD) & (RV > 220) then (PROPOSE
  (SEVERITY SEVERE .8))
```

Hypothesis element before rule evaluation:

```
hypo-element name: DISEASE-05
type: (OAD .8)
severity: (SEVERE .6)
```

Hypothesis element after rule evaluation:

```
hypo-element name: DISEASE-05
type: (OAD .8)
severity: (SEVERE .92) <-----
```

Rule Evaluators

The applicability of a rule is determined by how the premises in the LHS are evaluated. The term "applicability" can have a different definition for different problems--it may mean that all or some of the premises in the LHS need be TRUE, or that the premises require more complex evaluation. The user can define this "applicability" in the form of a function to serve as the "LHS evaluator." Some pre-programmed LHS evaluators are available; for example, a type of threshold evaluator used in MYCIN.

EXAMPLE 4: LHS Evaluator: \$ANDMIN (This function is similar to the one used in MYCIN where evidence with less than .2 value is ignored.)

Hypothesis element name: DISEASE-03

```
disease: (OAD .8)
subtype: (EMPHYSEMA .1)
severity: (SEVERE .6)
```

Rule 1: if (isa disease OAD) and
(isa subtype EMPHYSEMA) then...

Rule 2: if (isa disease OAD) and
(grdeg severity MODERATE) then...

Using \$ANDMIN, only Rule 2 is found applicable within the context of the given hypothesis state.

Example 5 shows how the features described thus far are represented within a KS. Note that some relevant features (e.g. credibility computation) reside outside the KS, because they are relevant to all the KSs in the user program.

Example 5: A KS Summary (AGE version of PUFF)

--Knowledge Source: OAD-SUBTYPERULES

Precondition: (OAD-DEGREE)

Inference levels: (from . to)
(DISEASE . DISEASE)

```
Links between levels: NONE
Hit strategy: (MULTIPLE)
Local variable bindings: NONE
[other relevant information
  LHS-evaluator: $ANDMIN
  Wt-adjuster: $ADJWT]
```

```
if
  (GREQDEG ($VALUE OAD DEGREE) MILD)
  (GREQDEG ($VALUE PATIENT DIF-DEF) MILD)
  (GREQ ($DATA TLC-BB:OBS/PRED) 110)
  (GREQ ($DATA RV/TLC:OBS/PRED) 10)
then PROPOSE
  change.type MODIFY
  hypo-element OAD
  attr-val/link (SUBTYPE EMPHYSEMA .9)
  (FINDINGS "OAD, diffusion defect, elevated
    TLC and RV together indicate Emphysema")
  event.type OAD-SUBTYPE
  comment (Rule 31 of MYCIN version)
```

3.1.3. Knowledge Utilization and Control Mechanisms

As mentioned earlier, the invoking of higher level knowledge about the appropriate use of problem solving methods and the invoking of other domain-specific knowledge appropriate to the situation, are both accomplished in the Control component. Until recently, very few attempts had been made to express control information explicitly. Davis [6] addressed the problem of expressing knowledge about other knowledge in production rule form and called it *meta-knowledge*. Nii [14] attempted to distinguish inference-generating KSs and knowledge-utilization KSs by organizing the KSs themselves into a hierarchy. In AGE, we have tried to provide concepts and mechanisms whereby users can express different types of higher level knowledge. The structure of the Control component is described below very briefly (see Appendix II for more detailed description). Whether the whole structure is sufficiently general to handle a variety of problems is still under study.

There are several system components grouped under the heading of *Control*. The various sub-components can be individually specified or programmed by the user. In order to simplify the designing process for the user, AGE provides pre-programmed components where appropriate. The control components that need to be specified are:

1. input data format,
2. initialization function,
3. processing method to be applied to each inference step,
4. rules or procedure to select the next step to be processed,
5. rules to select the relevant KS to process the selected step,
6. termination condition, and
7. post-processing function.

Components 1 and 2 deal with getting input data and performing setup operations in the user's program. The data can be brought in from a file or typed in from a terminal. The data elements are bound to names prespecified by the user. Within the initialization function the user may preprocess the data or perform other initialization activities--the only requirement for AGE is that the initialization function return a name of the first KS to be invoked.

Components 3-6, called the *control kernel* (Figure 2), constitute the core of the Blackboard control component. The primary functions of the control kernel are to select and invoke the appropriate KSs and to select the focus of attention within the blackboard. The control in the user's program is a simple loop:

1. **Rule evaluation:** In the invoked KS each rule is evaluated according to the user specified LHS-Evaluator (sec. 3.1.2). When the LHS meets the evaluation criteria, i.e. evaluates to True, the rule is *fired*.
2. **Inference generation:** Fired rules either (1) PROPOSE change in the hypothesis (this change is made in the hypothesis and is also placed on the Event-list), (2) indicate that some change in the hypothesis is EXPECTED to occur (this expectation is placed on the Expectation-list), or (3) indicate that the hypothesis needs to ACHIEVE a particular value or a state (this goal is placed on the Goal-list). Each of these actions is called a *Step*.
3. **Focus of attention:** Within the Step-selection module, select a step (an event, an expectation or a goal) and a hypothesis element or data to process next. Within the KS-selection module, choose a KS relevant to the selected step, and invoke that KS. (The KS may have associated with it a *processing method*; e.g., a KS may require a backward-chaining mechanism for the rules to achieve a goal.) [Go to 1]

Components 5 and 6 deal with terminating the kernel control loop and processing the completed hypotheses. One characteristic of the Blackboard model is that there is no prescribed way that the incremental hypothesis formation process terminates short of running out of relevant KSs. Thus, the user needs to specify the conditions under which the processing is to terminate; AGE monitors for the occurrence of these conditions. In the postprocessing function, the user may perform any further processing she wishes, for example, print the current hypothesis.

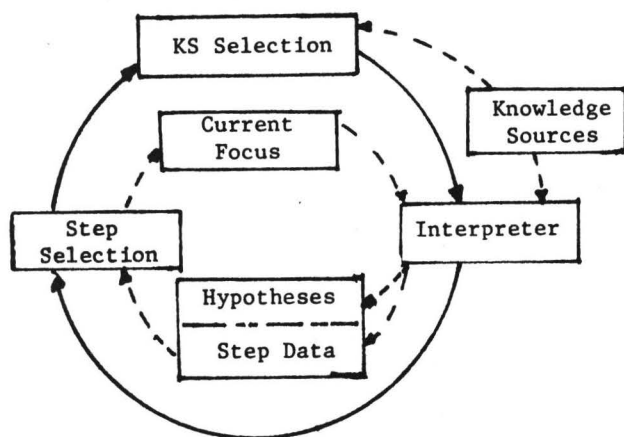


Figure 2. Control of Blackboard Model in AGE

Control Macros

Since the specifications required for the control kernel are quite complex, AGE provides control macros, currently one for *event-driven control* and one for *expectation-driven control*.

Event-Driven Macro: Event-driven hypothesis formation is characterized by incremental formation of hypothesis elements from evidence found in data or in lower level hypothesis elements. The elements can be processed by rules either on the basis of first-in-first-out, first-in-last-out, or best-first. These correspond roughly to breadth-, depth-, and best-first processing of the hypothesis space. When an element is chosen

to be processed, it is called a *focused element*. The event-type associated with the focused element helps determine which KS is to be invoked in the future.

The control loop for the Event-driven macro is:

1. <rule> modifies the hypothesis and causes an event (with associated event-type).
2. If <event-type> = <KS-precondition> then invoke the KS.
[Go back to 1]

Expectation-Driven Macro: Within an expectation-driven system, expectations in the hypotheses are generated by the rules. These rules are normally grouped around objects and their properties, similar to a schema (frame) organization. They usually form models of objects from which other properties can be inferred or can be expected to occur. For example, a rule of the form:

```

if (isa disease oad) & (isa severity severe)
then (expect RV >200)
    (expect RV/TLC.RATIO >20)
    etc.
  
```

has a schema-like flavor and produces expectations that need to be verified.

In order to determine if an expectation has been met, or can be met, the user must provide an *expectation matching function*. AGE always checks to see if the expected situation has occurred either in data or in the hypothesis (i.e. performs a passive match of expectation).

The control loop for the Expectation-driven macro is:

1. <rule> generates an expectation.
2. If an expectation is met, then modify the hypothesis as specified. This action generates an event.
3. If <event type> = <KS precondition> then invoke the KS.
[Go back to step 1]

For most problems, both the expectation-driven (or model based) and the event-driven (or data-driven) methods are needed in some combination. For the sake of completeness the control components from which the macros were created are described in Appendix II.

3.2. The Intelligent Front-end in AGE

AGE assumes that the user neither knows nor understands the concepts and implementations of the various program components described above. It is the task of the front-end to guide the user in constructing a program using the component parts. Currently the *intelligence* in the front-end is limited to: (a) a tutor subsystem that allows the user to browse through the textual knowledge base, and (b) a design subsystem that guides the user through each step of program specification. An "unfamiliar" user is always introduced to AGE by way of the tutor subsystem.

The textual knowledge base contains (a) a general description of the building-block components at the conceptual level, (b) a description of the implementation of these concepts within AGE, (c) a description of how these components are to be used within the user's program, (d) how they can be constructed by the user, and (e) various examples. The information is organized in a network to represent the conceptual hierarchy of the components and to represent the functional relationship among them.

The design subsystem guides the user in *design and*

construction. The knowledge necessary for AGE to accomplish this task is represented in a data structure in the form of an AND/OR tree that, on one hand, represents all the possible structures available in the current AGE system; and, on the other hand, represents the decisions the user must make in order to design her program. Using this schema, the design subsystem can guide the user (in the *guided design mode*) from one design decision point to another. At each decision point, the user has access to the textual knowledge base, to advice on the decisions to be made at that point, and to acquisition functions that aid the user in specifying the appropriate component. The user also has direct access (in the *unguided design mode*) to various acquisition and editing functions.

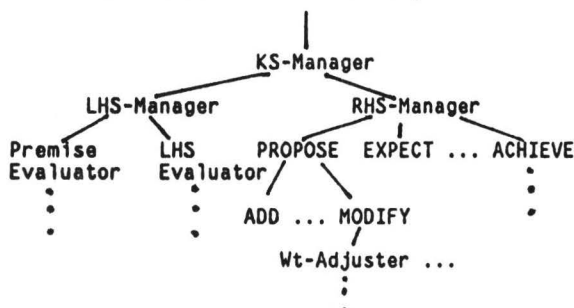
The reader is referred to Appendix I which contains extensive examples of the various interactions currently possible in AGE.

4. DECOMPOSITION OF FRAMEWORKS

Although the intention of this paper does not include discussion of the architecture of AGE itself, we briefly describe some of the motivation behind the decomposition of problem-solving frameworks into interchangeable parts.

Our concept of a software laboratory is a facility in which the users are provided with a variety of *preprogrammed problem-solving frameworks*--similar in spirit to designs of prefabricated houses. The user augments and modifies a framework to develop her own programs. In order to provide such a facility, a framework must be built with parts that can be "unplugged" and replaced. Each framework is decomposed into self-contained modules, creating a hierarchically organized set of component parts. These parts in turn are interpreted or evaluated by "various" managers. As an example, a part of the decomposed Blackboard framework is shown below:

EXAMPLE 6: Blackboard Model Decomposition



Theoretically, then, each module (with all of its subparts) is replaceable. For example, the whole RHS-Manager can be replaced if the user wants a definition of the RHS of rules different from the one we currently provide.

However, complete modularity is not achieved, because some of the modules are not independent. From a practical standpoint the greatest barrier to replacing some of the modules is in their inaccessibility. Nevertheless, as in prefabricated houses, every module can be modified or replaced--some changes are more costly than others in terms of time and detailed knowledge of the implementation needed. In general, those parts nearer the bottom of the hierarchy are more accessible and easier to change. For example, the definition of ADD (add a new hypothesis element) can be changed by simply redefining the ADD function. Or, another function that makes changes to the

hypotheses in a different way can be added. On the other hand, the LHS- and the RHS-Managers are not independent. For those components that are independent, and easily modified or replaced, AGE provides some aids for making changes--acquisition functions and descriptions in the textual knowledge base. For those more difficult to change, we provide less aid with the intention of discouraging novice users.

In order to have a useful software laboratory, we need to provide the user with diverse tools--some tools need to be flexible in their utility while others need to be fine-tuned for specific use; and some tools require substantial skill in their use while others do not. By organizing the program modules by their utility and providing easy access to those that are general and easy to use (but allowing skilled users access to all modules), we hope that the laboratory can be useful to a wide range of users.

5. SUMMARY AND CONCLUDING REMARKS

AGE is an experimental program currently running on PDP-10 at the SUMEX-AIM Computing Facility at Stanford University. The immediate goal of the project is to provide software tools with which a knowledge engineer can reduce the time it takes to build knowledge-based programs. The longer range goal is to aid people less knowledgeable in AI methods to build knowledge-based programs. We have taken a small step toward reaching these goals by exploring the various ways in which Blackboard-based programs can be built, and by implementing a set of parts from which such a program can be built.

In the process of building AGE, we have used it to write some programs: CRYPTO, a program that solves cryptogram problems; a portion of a bidding strategy problem in bridge games; two different versions of PUFF [9,12]--one using the Event-driven control macro and another using the Expectation-driven control macro [15]. Since the domain-specific knowledge for PUFF already existed and was being used in EMYCIN, the AGE version took about a week to bring up--time to reorganize the existing rules into KSs and to rewrite the rules in AGE rule syntax. Currently, the CRYSLIS program [7] is being rewritten using AGE.

To the extent that we have been able to use AGE to develop some programs, we feel encouraged to continue with further development. However, there are still many issues that need to be explored in the current work--the adequacy of the user interface and debugging facility; generality of the current functional breakdown of the Blackboard model; sufficiency of the current rule syntax and semantics to express a wide range of knowledge; adequacy of the production rule representation itself for a variety of tasks; reliability of the system; etc. We have made no progress in providing a facility for explanation within the object program; we need more basic research in this area before such a facility can be implemented.

APPENDIX I: Building Knowledge-Based Program with AGE

Introduction

This appendix shows the AGE system from an user's point of view. We have included an example program with parts of actual protocols to illustrate various facilities.

Cryptograms

The problem used as an example is translating cryptograms. The cryptogram solution consists of a four level hierarchical

hypothesis structure. The rules to solve cryptograms were provided by a human "expert". The task for CRYPTO is to take a "newspaper" cryptogram (10 to 20 words long, about 100 letters) and using a set of heuristics, translate it into an English sentence. For example, the cryptogram "GSHSUO CI DXDV BSS CLESVBFG B BS HD NCGU HJB ISLD JGNCGU EDSEYD EVSX D BZFB BZDVD'I IBCVY F RDP SR JI PZS FVD BSS ILFYY." translates to "Nobody is ever too important to be kind but some unkind people prove that there's still a few of us who are too small." The word boundaries and punctuation are given. There is a one-to-one correspondence between the cryptletters and their translation, and no letter translates to itself.

Typical run

An AGE user typically builds a knowledge-based program over a period of time, conducting many hands-on sessions. In the first session the user gets acquainted with the system, learning the different components available and the system conventions, and discovering what information she must provide (i.e., what to think about before the next session). During the next few sessions the user designs and builds her program. Finally, the user iterates on checking out the program, running it, and modifying it.

TUTOR: The TUTOR subsystem is described in Section 3.2.

[after the user logs on, AGE begins by determining if the user needs a tutorial]

Are you already familiar with this system?: No

This and all subsequent sessions will be recorded.

The record will go on your directory as DRIB<name><date>.

Following is a list of AGE conventions which may be useful.
[description of AGE user conventions]

[canned text on other topics]

What follows is a tutorial session to acquaint you with AGE.

[The following text describes the organization of the AGE on-line manual]

Each topic in the AGE system has one or more modifiers attach to it. A modifier may be a descriptive word (e.g. Hypothesis.-element) or it may be one of the following:

ISA: Definition or definitional description.

FORMIS: Representation or structural description.

FORMEDBY: How <topic> is generated or created internally during the user program execution.

TOSPECIFY: Description of the acquisition function.

GEN: generally accepted description.

AGE: definition or description as interpreted for AGE.

[canned text on further topics]

[The following interaction shows cross-referencing]

---There are some pre-assembled control structures available for convenience. (See CONTROL.MACRO)

References are: Hypothesis.structure Control.macro

Do you want to see them now?: y

HYPOTHESIS.STRUCTURE Y/N:: y

The hypothesis structure in AGE is hierarchically organized. There may be more than one hierarchy in the hypothesis (generally known as blackboard planes). Each hierarchy

consists of linked hypothesis elements, HYPOTHESIS.ELEMENT, grouped into levels of abstractions.

[Tutorial session is eventually terminated.]

This ends the AGE tutorial session. From this point on you will be given a menu of tasks to select from. To start, I suggest you begin with the DESIGN task.

DESIGN: Refer to section 3.2 for a detailed description of the Design subsystem.

What do you want to do next?

Tasks available are: [These are top level tasks in AGE]

name	description
----	-----
Tutor	System explanation.
Browse	Browse thru the AGE manual.
Design	Create the user program.
SAVE	Save the user program on a file.
Getsys	Get the user program from a file.
Check	Check syntax of the user program.
Run	Execute the user program.
SYSout	Save everything (ie. SYSOUT)

:: design

Create the user program.

What is the name of your system?: CRYPTO

[AGE determines if there already exists a program by that name]

Do you want a Guided design session, or Unguided session?

:: g

[Only a minimal number of consecutive letters that uniquely identifies the prompt is needed.]

Beginning DESIGN and ACQUISITION: You will need to design and specify three major components of your program: the hypothesis structure, the domain-specific rules, and some control information.

You may begin working on : Hypothesis.Structure Knowledge.domain-rules Control

During this design session, in addition to specifying or editing components of the system, you may also ask for ADVICE, more INFORMATION from the manual, or to QUIT the session.

What do you want to do next?

Your options are: Hypothesis.structure Knowledge.domain-rules Control Advice Information Quit

Type H/K/C/U/A/I/Q: a [asking for advice]

Specify the hypothesis structure first; it will aid you in structuring your problem. Then enter some domain rules, followed by control information. Within the control specification, choose system-provided control methods to see how your rules work. More rules can be added or more sophisticated control can be specified later.

HYPOTHESIS: See Section 3.1.1 for a general discussion of the Hypothesis.

CRYPTO uses a four-level structure, as shown in Figure A1. At the highest level (SENTENCE) there is one named node CRYPTOGRAM, where information inferred about the whole cryptogram will be stored. At the WORD level, elements WORD1 to WORDn are created dynamically for each word of the cryptogram; and at the LETTER level, elements LETTER1 to

LETTER1 to LETTERm for each letter.

levels

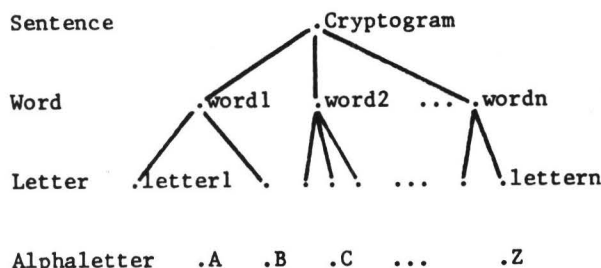


Figure A1. CRYPTO Hypothesis Structure

What follows is an example protocol of the acquisition of the hypothesis structure for CRYPTO.

What do you want to do next?

::Info [ask for information]

Select a topic.

::hypothesis.tospecify

Function: (HYPO)

The user is asked for the names of levels of hypothesis elements, and for the names of the attributes at each level. The level names must be unique; the attribute names need only be unique within a level. When AGE asks for attributes, only attributes should be given and not the names of the links between two levels. AGE will start from the highest level, LEVEL1.

Do you want to work on Hypothesis.structure?: y

Aquiring HYPOTHESIS.STRUCTURE

Name of LEVEL1 ?

A level name can be any literal atom not already used for the name of another level. Type Done to quit.

:: sentence

Attribute?

Attributes must be unique within a level.

:: words

Attribute?

[etc.]

Name of LEVEL2 ?

:: wurd [misspelled name--to be corrected later]

Attributes?

:: letters

[The remainder of hypothesis acquisition and editing deleted.]

:: print [print the hypothesis structure]

LEVEL1 - SENTENCE

WORDS - NIL

ENGLISHWORDS - NIL

LETTERS - NIL

LEVEL2 - WURD [misspelled level name])

LETTERS - NIL

ENGLISHWORD - NIL

ORDREINSENTENCE - NIL [misspelled]

POSSIBLEWORDS - NIL

LEVEL3 - LETTER

CRYPTLETTER - NIL

ORDERINWORD - NIL

RELFREQ? - NIL

KSs: Refer to Section 3.1.2 for a general description of the KSs

as used in AGE. The protocol shows the partial acquisition of two KSs. The specification of each KS begins with some declarative information needed to evaluate the rules in the KS. Then the user is prompted for parts of the rules. The first KS is a set of rules that look at the whole cryptogram. It is called once after initialization and any time thereafter to find a new part of the cryptogram to focus on. This KS assumes that certain information already exists in the hypothesis structure as a result of initialization. The first rule looks for one-letter words which could either be A or I, with A more likely than I.

You need to specify: Rule.evaluator Knowledge.sources Control

You may modify: Hypothesis.structure

[AGE keeps track of what has and has not been specified]

What do you want to do next?

:: kn

What is the name of the KS you want to create or add rules to?

:: wholesentencerules

[New knowledge source]

What events invoke this knowledge source?

:: always [Always means any event can be a precondition to this knowledge source]

Between which Hypothesis levels does this KS make inferences?

From?

:: letter

To?

:: alphaletter

From?

::none

Which links are used to record inferences made by this KS?

:: possiblevalue [Possiblevalue links will connect elements from the letter level to possible values in the alphaletter level]

Inverse link?

:: possiblevalue/of

Link?

:: assignedvalue [Assignedvalue links will connect positively assigned values from the alphaletters to the cryptletters on the letter level]

Inverse link?

[etc.]

Do you want to use multiple or single hit strategy for this KS?

You can specify Onceonly with Single or Multiple.

:: single onceonly [Single to fire only one rule and Onceonly to mark that rule so that it won't be fired again]

Local variables can be defined as (variable value) or (variable).

Define a local variable.

:: (wordnodes (\$value 'cryptogram part))

[Store the node names for the word nodes in WORDNODES.

The function \$VALUE retrieves those names by following the "part" link from the CRYPTOGRAM node]

Define a local variable.

:: (word)

Define a local variable.

[etc.]

What is the first condition? Type None when done.

[This problem involves searches which are awkward with the current rule syntax and requires extensive Lisp code. See Example 5 for the more usual use of rules.]

:: (setq word (for wd in wordnodes thereis
(and (eg (length (\$VALUE wd letters)) 1)

(null (\$value wd englishword])
[looking for a word of length 1, not already assigned a value]
 Next condition?
 :: none

Next, you will be prompted for the RHS of the rule.
 PROPOSE, EXPECT, GOAL, or LOOP?
 ::? propose *[ask about PROPOSE]*
 PROPOSE is used to change the hypothesis structure.
 :: prop *[specification of PROPOSE follows]*
 ch.type:: supersede *[replace current value]*
 hypo-element:: word *[of word element]*
 attr-value:: (englishword 'a) *[with "a"]*
 link-node:: none
 event.type:: assignword *[name of the event it generates]*
 comment:: (a one letter word is most likely 'a')

PROPOSE, EXPECT, GOAL, or LOOP?
 :: propose *[more than one action allowed in the RHS]*
[etc.]
 link-node:: (assignedvalue 'a) *[change the relational link]*
 event.type:: assignletter
 comment:: (assign the letter 'a' to the cryptletter)

PROPOSE, EXPECT, GOAL, or LOOP?
 :: none
 More rules?: no
 What is the name of the next KS you want?
 :: fill-inrules
[New knowledge source]
[some declarative information deleted]
 PROPOSE, EXPECT, GOAL, or LOOP?
 ::? loop *[ask about LOOP]*
 LOOP is used to indicate an iterative action, either PROPOSE, EXPECT, or GOAL. If LOOP is used, you will be prompted for the name of an iteration variable and the name of a list of values on which to repeat the action.
 :: loop
 Iteration variable?
 :: ol
 Name of the list to be iterated on?
 :: otherletters
 PROPOSE, EXPECT, or GOAL?
 :: propose
[remainder of rule acquisition delete]

CONTROL: Refer to Section 4 and Appendix 11 for discussion of knowledge utilization and control mechanisms. The overall strategy for solving cryptograms is to focus on a particular word or a combination of letters that has "potential"; work on the focussed item and follow through on all the ramifications, which in turn may generate a new focus.

:: control

Control specification: Currently there are three methods of specifying the control information. Two are macros which require minimal specification, EVENT-DRIVEN-MACRO and EXPECTATION-DRIVEN-MACRO. The third method, DETAILCONTROL, involves detailed specifications of several subcomponents.

New additions to design component: (Detailcontrol Control.macro. Event-driven Control.macro.expectation-driven)

[When a node is chosen, it opens other nodes that require further specification]

What do you want to do next?

:: detail

Detailed Control Specification: There are several components grouped under the heading of DETAILCONTROL. To simplify the designing process, AGE provides default specification where appropriate. The components that need to be specified are:

1. input data format: INPUTFORMAT,
2. initialization function: INITIALIZATION,
3. processing method to be applied to each step-type,
4. rules to select a step to be used as the current focus,
5. rules to select relevant KS (or procedure) to process the selected step: (3,4,5 part of KERNEL),
6. termination condition: TERMINATION, and
7. post processing function: POSTPROCESSING.

New addition to design component: (Inputformat Initialization Control.Kernel Termination Postprocessing)

What do you want to do next?

:: inputform

The input data you use in your system will be stored on an implicit level of the hypothesis structure called INPUTDATA. A function \$DATA is available to access the data. At the beginning of each run you will be asked to input (from a file or TTY:) the actual data associated with each data name.

First name?

:: inputcryptogram

Next name?

:: done

What do you want to do next?

:: initial

You must provide a function that will return, as its value, the name of the first KS to be invoked. Within this function you may do any processing you wish.

[NEW FUNCTION]

What would you like to call your initialization function?

:: crypto-initialization

Your initialization function must return the name of a KS.

Please define CRYPTO-INITIALIZATION

(CRYPTO-INITIALIZATION (LAMBDA NIL

[This line provided by AGE]

(prog (cg)
 (cg + (\$data inputcryptogram))
 (fill-inlevels)
 (computerelfreqs)
 (return 'wholesentencerules])

What do you want to do next?

:: terminate

You must provide a function which will determine when to terminate your program.

What is the name of your termination function?

:: crypto-quit

[define CRYPTO-QUIT function in the same manner as above]

What do you want to do next?

:: post

you can provide functions for printing the results, translating the results into a more readable form.

Take up this option or bypass it? [optional component]
:: take
What would you like to call your postprocessing function?
:: decodedprint
[Define DECODEPRINT]

What do you want to do next?
:: control.k
This function allows the user to specify various options which determine how events, expectations, and goals will be processed. First we will look at the event processing information.

Do you want to enter kernel information now?: y
Which event selection method would you like to use? LIFO and FIFO are available or you can write your own function. Type LIFO, FIFO, or function name.
:: ?
The Event-selection is used to decide which event from the event list to process next. LIFO chooses the last event added. FIFO chooses the earliest event. You can provide your own function. For example, bestfirst might depend on the weights associated with each event.
:: lifo

Name an event type to be collected.
:: fill-in
:: done

The control rules are used to determine what step type to process next. Each control rule consists of a condition and a step type.

STEPTYPE:: event
CONDITION:: expectationlist
STEPTYPE:: expectation
CONDITION:: none

USER FUNCTIONS: The final information the user must provide is the names and definitions of the functions used in the rules or called from other user-defined functions. The user can name and define functions one at a time or load them from files.

EDITING: There are two ways to edit: (1) by calling the appropriate editing function from Unguided design session or (2) during the Check-out session. The Check subsystem described below.

SYSTEM CHECKOUT: The Check subsystem does a syntax check on the user created program. It finds errors in the designed structures and points out missing, but required, information. Check also allows the user to edit the errors as they are found by calling the appropriate editor.

What do you want to do next? : check
Check syntax of the user program.
checking hypothesis levels - ALPHALETTER ...
checking knowledge sources -FILL-INRULES ...
Warning: 2 events not used as preconditions
 ASSIGNWORD FILL-IN
[they will be used by KSs to be defined later]
1 syntax errors in the user program.
Do you want to edit the errors now?: n

RUNNING: Once all the information is specified, the user can run her program in two modes, debugging mode or normal mode. In debugging mode, AGE prints a trace of the control flow showing the rules evaluated, the actions taken, and the steps and focuses selected. In normal mode, AGE only prints the input data and the results.

A complete run will begin with a request for input data. Next the user's initialization function which returns a KS name is evaluated. That KS is the first to be interpreted and starts the control loop, which will continue until the termination condition is met. Finally, the postprocessing function is called. The following is a sample run of the CRYPTO program. We have deleted a large section of the protocol to save space.

What do you want to do next?: run [top level prompt]
Execute the user program.
Do you want to input data from a file or from a terminal?
(filename or TTY::: crypto-datal [input from a file])
((INPUTCRYPTOGRAM (GSHSUO CI DXOV BSS CLESVBFG B BS
HD NCGU HJB ISLD JGNCGU EDSEYD EVSXD BZFB BZDVD'I IB
RDP SR JI PZS FVD BSS ILFYY)))

Knowledge source invoked - WHOLESENTENCERULES
[WHOLESENTENCERULES is the KS name returned by the initialization function.]

Left hand side of applicable rule -
((SETQ WORD (for WD in WORDNODES thereis (AND ...
[first rule to fire, word17, letter73, is probably "A".
Two events result, ASSIGNLETTER and ASSIGNWORD])

NAME OF NEW EVENT: ASSIGNWORD
NAME OF NEW EVENT: ASSIGNLETTER

NEXT STEP: EVENT [choose the next step - EVENT]

FOCUS EVENT-(ASSIGNLETTER(LETTER73)(ASSIGNEDVALUE A
WHOLESENTENCERULES EV2 NIL)
[Since the ASSIGNLETTER event was added to the
EVENTLIST last, it is the next focus event under the depthfirst
(LIFO) selection method. When there are no further ramifications
of ASSIGNEDLETTER, ASSIGNWORD will become
the focus event.]

Knowledge source invoked - FILL-INRULES
Left hand side of applicable rule -
((SETQ OTHERLETTERS (for L in CRYPTLETTERNODES join
[find other letters with same cryptvalue as letter73
FILL-IN those letters with value "A"]

NAME OF NEW EVENT: FILL-IN

NEXT STEP: EVENT

FOCUS EVENT - (FILL-IN (LETTER92) (ASSIGNEDVALUE A)
(ASSIGNEDVALUE A) EV13 NIL)
[letter92 was assigned last, thus with a depthfirst selection
method, it becomes the new focus.]
[etc.]
[The rest of the protocol continues in the same manner until
terminated.]

Summary

One common question asked about the CRYPTO system is, "When CRYPTO makes an incorrect assignment, can CRYPTO back up?" The answer is no. There is no back-up or alternative hypothesis capability. In CRYPTO we believe

that with enough rules and with the POSSIBLEVALUE attributes and weights, the system will always make the correct assignments. This belief stems from the sessions recorded with the expert newspaper cryptographer, who never needed to "back up." The expert never guessed at a value or tried to confirm or contradict a guess by looking at its ramifications. However, we do recognize that the concept of alternative hypotheses is a powerful one, and we expect to add facilities to AGE to allow them in the future.

APPENDIX II: SUMMARY OF THE CONTROL KERNEL

The Control Kernel consists of several components (Figure B1) each of which must be specified by the user. The sequence of actions which are taken within the Kernel is described below.

1. Invoke rules (CONTROLRULES) to select the next step-type to process--EVENT, EXPECTation, or GOAL;
2. If the selected step-type is EVENT, then
 - (a) Select an event, (SELECTIONMETHOD provided by the user);
 - (b) Collapse the EVENTLIST by deleting/merging events referring to identical attributes and values of the events specified by the user (COLLECTIONRULES);
 - (c) Select a KS (EVENTRULES generated by AGE from the KS preconditions).
3. If the selected step type is a goal (ACHIEVE), then
 - (a) Select a goal (SELECTIONMETHOD);
 - (b) Evaluate to determine if the goal has been met (MATCHER).
 - (b) If not, apply a goal-seeking method (SEEKMETHOD).
 - (d) If the goal has been met, execute the remainder of the RHS of the rule that generated the goal.
4. If the selected step-type is EXPECT, then
 - (a) Select an expectation (SELECTIONMETHOD);
 - (b) Evaluate to determine if the expectation has been met (MATCHER),
 - (c) If so, process the expected event by executing the remainder of the RHS that generated the expectation.

Step-type	Event	Expectation	Goal
SELECTION METHOD	FIFO LIFO <fns name>	FIFO LIFO <fns name>	FIFO LIFO <fns name>
COLLECTION RULES	<KS name> ()	---	---
MATCHER	[EVENTRULES]*	<fn name> passivematch	<fn name> nii
SEEKMETHOD	---	---	<fn name>

Figure B1. Summary of Control Kernel Components and Options

REFERENCES

- [2] Bennett, J., Creary, L., Engelmores, R., Melosh, R., *SACON: A knowledge-based consultant for structural analysis*, Heuristic Programming Project Memo HPP-78-23, 1978.
- [3] Bloom, B.S., *Taxonomy of Educational Objectives*, McKay, New York, 1956.
- [4] Bobrow, D. G. and Winograd, T., *An Overview of KRL, a Knowledge Representation Language*, Cognitive Science, vol 1, no 1, pp3-46, 1977.
- [5] Davis, R. and King J., *An overview of production systems*, Machine Intelligence 8: Machine Representation of Knowledge, Elcock, E.W. and Michie, D. (eds.), John Wiley, 1977.
- [6] Davis R. and Buchanan B.G., *Meta-level knowledge: overview and applications*, Proc. IJCAI 5, 1977, pp.920-927.
- [7] Engelmores, R.S. and Nii, H.P., *A knowledge-based system for the interpretation of protein x-ray crystallographic data*, Heuristic Programming Project Memo HPP-77-2, January, 1977.
- [8] Erman, L.D. and Lesser, V.R., *A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge*, Proc. 4th IJCAI, 1975, pp.483-490.
- [9] Feigenbaum, Edward A., *The art of artificial intelligence: I. Themes and case studies of knowledge engineering*, Proc. IJCAI 5, 1977, pp.1014-1029.
- [10] Hayes-Roth, Frederick and Lesser, Victor R., *Focus of attention in a distributed-logic speech understanding system*, Proc. of IEEE Int. Conference on ASSP, Philadelphia, 1976.
- [11] Hempel, Carl G., *Philosophy of Natural Science*, Foundations of Philosophy Series, Prentice-Hall, Inc., 1966.
- [12] Kunz, J.C., Fallat, R.J., McClung, D.H., Osborn, J.J., Votteri, B.A., Nii, H.P., Aikins, J.S., Fagan, L.M., Feigenbaum, E.A., *A physiological rule based system for interpreting pulmonary function test results*, Heuristic Programming Project Memo HPP-78-20, (submitted to Computers and Biomedical Research), 1978.
- [13] Lesser, V.R. and Erman, L.D., *A retrospective view of the HEARSAY-II architecture*, Proc. 5th IJCAI, 1977, pp. 790-800.
- [14] Nii, H. P. and Feigenbaum, E.A., *Rule-based understanding of signals*, Pattern-Directed Inference Systems, D.A. Waterman and R. Hayes-Roth (eds.), Academic Press, 1978.
- [15] Nii, H.P. and Aiello, N., *AGE (Attempt to Generalize): Profile of the AGE-0 System*, Stanford Heuristic Programming Project Memo HPP-78-5 (Working paper), June 1978.
- [16] Shortliffe, Edward H., *Computer-Based Medical Consultation: MYCIN*, American Elsevier, New York, 1976.
- [17] Sridharan, N.S., *AIMDS User Manual*, Report CBM-TR-89, Department of Computer Science, Rutgers University, June, 1978.
- [18] Stefik, Mark, *An examination of a frame-structure representation system*, Stanford Heuristic Programming Project Memo HPP-78-13, September, 1978.
- [19] Teitelman, W., et al, *Interlisp Reference Manual*, Xerox Palo Alto Research Center, October, 1978.