FRAMENTEC

# S.1

KNOWLEDGE ENGINEERING FOR PRODUCTIVITY

S.1

# Overview

S.1 is the most powerful Knowledge Engineering tool available today. It integrates state of the art technology with the experience gained from building more than two dozen expert systems.

S.1 is a tool specifically designed to work in an industrial environment as opposed to an applied research tool.

S.1 is an integrated package of software, training and support, designed to assist knowledge engineers in the development of large scale expert systems.

S.1 derives its power by integrating strategies for representation, inference, and control. An S.1 knowledge base is composed of three kinds of knowledge: factual, judgmental, and procedural. Each kind of knowledge is encoded separately. Factual knowledge, representing assertions about objects and relationships between objects, is encoded in a specialized **"frame system"**. Judgmental knowledge, representing heuristics and rules of thumb about the application, is encoded in **"if-then" rules.** Procedural knowledge, representing sequences of problem-solving steps, is encoded in **"control blocks"**.

S.1 provides the means for the knowledge engineer to solve complex reasoning problems that may require substantial quantities of factual, judgmental, and procedural knowledge. Distinguishing these different types of knowledge permits the efficient implementation, maintenance and use of large-scale expert systems that can be maintained, updated, and used effectively.

In addition to software, a two-week training course for S.1 is available, covering the features and use of the software, extensive documentation, a set of sample expert systems, product support service, and access to Framentec's applications engineering services.

# Architecture

S.1 provides facilities for knowledge engineers (people who design and implement expert systems) and for users (people who consult with expert systems to solve a particular problem).

The knowledge engineer constructs a knowledge base using an editor. When the editor is exited, the knowledge base can be loaded into S.1 and checked for syntax errors and internal consistency. The knowledge engineer can list the knowledge base (or portions of it), including cross-reference information that is helpful in verifying the accuracy of the knowledge base.

To test the knowledge base, the knowledge engineer can run consultations, save and rerun test cases, trace the system's actions, break at specified kinds of events to explore the dynamic environment, and keep a typescript of the consultation for later examination.

The goal of an S.1 consultation is to complete execution of a **top level** control block. This control block structures the consultation to achieve the desired result, which is normally to establish one or more conclusions and print these for the user. In the course of executing, the top level control block may in turn apply rules, examine facts, or execute other control blocks, as appropriate.

When the knowledge base is developed, the knowledge engineer can prepare a "consultation only" system for users. This system can protect the knowledge base from modification or disclosure by unauthorized persons.

The user can perform a consultation by responding to questions either by typing on the keyboard or pointing to options with a mouse. During or after the consultation, the user can request explanations of what, why and how conclusions were made.

# Documentation and Sample Expert Systems

The documentation for S.1 includes a reference manual, user's guide, training materials, and a library of sample expert systems.

- The S.1 Reference Manual contains an explanation of S.1's features and functions. This document defines the behavior of S.1 in detail.
- The S.1 User's Guide explains how to use S.1 to build expert systems. It describes techniques for using S.1 to represent and reason with knowledge.
- Training materials also serve as documentation. These include visual materials used during lectures, scripts for introductory exercises, advanced exercises, and partial solutions.
- The purpose of the Sample Expert Systems Library is to provide methodological guidance for users of S.1. The library contains two kinds of material. First, there is a guide to building expert systems. This aids in selecting an appropriate application, designing the expert system, managing the project, and documenting the result. The second part of the library details a specific expert system development project. This includes a final project report, a design summary, a knowledge base listing, demonstration cases, and an historical description of the evolution of the system with major design changes and their rationale.
- The sample expert system described in the library is included with the S.1 software. This system, with definitions of approximately 150 knowledge base objects, provides hands-on experience with a working knowledge system.

# Product Maintenance and Support

S.1 maintenance includes a telephone hotline to assist in product installation and product operation. S.1 purchasers also have access to Framentec's applications engineering service, which is not available to the general public. This service assists customers in designing and implementing their own applications. Applications engineering service is provided on a separate contract basis.

# Knowledge Engineering Features

**Multi-window development environment.**

To facilitate knowledge base debugging, the knowledge engineer's interface is organized into separate windows, each serving a specific function. A "typescript" window contains the actual consultation questions, answers, explanations, and conclusions. A "help" window describes the context and the current commands and options. A "function-key" window allows commands to be entered by pointing with a mouse rather than typing. A "Lisp window" provides the knowledge engineer with access to Lisp.

**Multi-level tracing and breaking.**

Different types of events can be selectively enabled for tracing or breaking during a consultation.

**Consistency checking.**

S.1 performs strong type checking to ensure consistent use of knowledge base objects.

**Saved cases and transcripts.**

To facilitate debugging, the knowledge engineer can store the answers from a consultation in a special "test case" file for later use. When running a consultation, input can be taken from this file, rather than the keyboard. An additional option

S.1

**Creation of consultation-only systems.**

The knowledge engineer can prepare a consultation-only system that provides a controlled environment for the users, protects the knowledge base from modification, and prevents disclosure of sensitive information. Two basic levels of protection are provided: changes to the knowledge base can be prohibited, and display of the knowledge base (including explanations) can be restricted.

**Dynamic creation of objects during consultations.**

A knowledge base contains descriptions of classes, which can be dynamically instantiated during a consultation to create multiple objects. For example, a consultation regarding rooms in a house might create each room as it is discussed or needed for the consultation. Relationships can be defined to link objects into structures; these relationships can be used in the reasoning process.

**Interconnection to external systems.**

Data structures can be passed between S.1 and Lisp procedures, providing an interface to external systems, such as sensors, data bases, and communications networks. In particular, S.1 permits access to existing software written in Fortran, C, and Pascal.

# S.1 – Selecting Applications

S.1 is best suited to a particular kind of knowledge engineering problem, known as **structured selection.**

Structured selection problems are characterized by a finite set of possible solutions, and the use of a structured reasoning process to identify one or more of the possible solutions for a given case. The steps in the reasoning process typically involve collecting, aggregating and abstrating information about the case, relating this abstract description to a general set of solutions, and then selecting and refining one or more specific conclusions.

Examples of structured selection problems include identifying a machine part failure, diagnosing an illness, recommending an appropriate tool for some task and advising on the options offered by a complex computer program.

# S.1 - Technical Review

The following sections present a technical overview of S.1, drawing some examples from the domain of the Framentec "Car Charging Advisor" sample expert system. This system helps an apprentice mechanic diagnose and correct problems with a car's charging system.

## REPRESENTING FACTUAL KNOWLEDGE

Factual knowledge in S.1 is represented as an extension of "object-attribute-value" triples. For example, "The cause of the car's problem is that the fanbelt is slipping", could be represented by identifying "the car" as the central object being discussed, "cause of problem" as the attribute, and "fanbelt slipping" as the value of that attribute. In S.1's language, this set of facts about the world is represented as follows:

**cause.of.problem[CAR] = fanbelt.slipping.**

Objects do not need to be explicitly encoded in advance of a consultation. Rather, they are created as needed by "instantiating" a more general structure, called a "class". As the name implies, a class is a description of a type of object. Individual objects, also called "instances" of classes, are established when they are required.

conditions involving boolean, numeric, or symbolic values and can lead to multiple conclusions.

### Procedural knowledge representation.

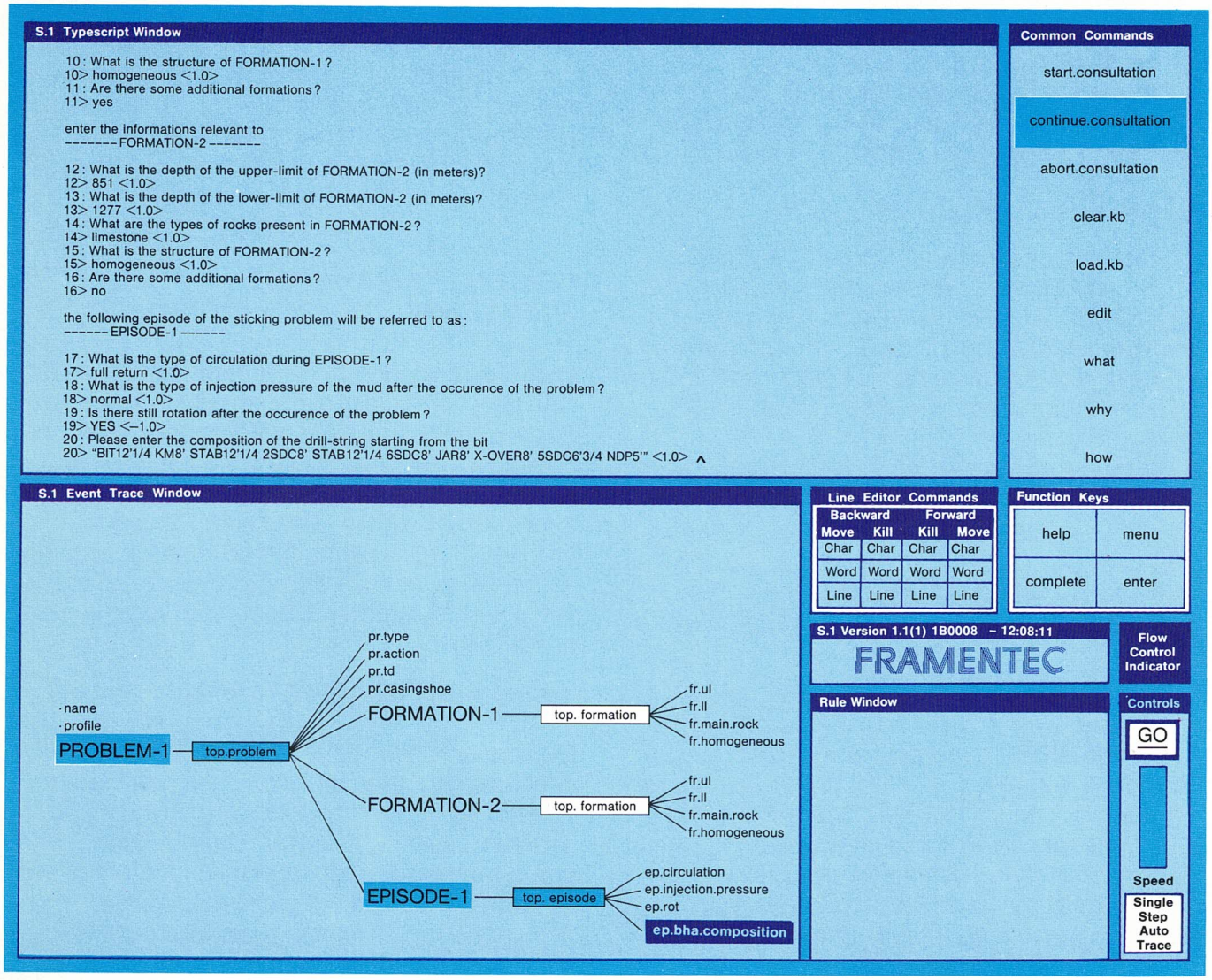
S.1 incorporates a procedural language for establishing the order and context in which rules are applied, and for altering the system's basic backchaining control strategy. This language eliminates the need to encode procedural knowledge in a declarative form or resort to linking external subroutines that augment the control strategy.

### Certainty factors.

Facts and rules can be modified by certainty factors, associating a degree of certainty or belief with each proposition or inference.

Certainty factors can be either positive or negative, so that evidence can be accumulated for or against hypotheses. A built-in certainty factor calculus is used to combine values. The conditions in rules can use a variety of predicates to test the degree of certainty associated with an hypothesis.

**The S.1 Development Environment** - Different windows allow the Knowledge Engineer to consult with the system (upper left), examine the reasoning tree (lower left), and enter into lisp (lower right). In this example, S.1 diagnoses a sticking problem on an oil well.



S.1

**Creation of consultation-only systems.**

The knowledge engineer can prepare a consultation-only system that provides a controlled environment for the users, protects the knowledge base from modification, and prevents disclosure of sensitive information. Two basic levels of protection are provided: changes to the knowledge base can be prohibited, and display of the knowledge base (including explanations) can be restricted.

**Dynamic creation of objects during consultations.**

A knowledge base contains descriptions of classes, which can be dynamically instantiated during a consultation to create multiple objects. For example, a consultation regarding rooms in a house might create each room as it is discussed or needed for the consultation. Relationships can be defined to link objects into structures; these relationships can be used in the reasoning process.

**Interconnection to external systems.**

Data structures can be passed between S.1 and Lisp procedures, providing an interface to external systems, such as sensors, data bases, and communications networks. In particular, S.1 permits access to existing software written in Fortran, C, and Pascal.

# S.1 – Selecting Applications

S.1 is best suited to a particular kind of knowledge engineering problem, known as **structured selection.**

Structured selection problems are characterized by a finite set of possible solutions, and the use of a structured reasoning process to identify one or more of the possible solutions for a given case. The steps in the reasoning process typically involve collecting, aggregating and abstrating information about the case, relating this abstract description to a general set of solutions, and then selecting and refining one or more specific conclusions.

Examples of structured selection problems include identifying a machine part failure, diagnosing an illness, recommending an appropriate tool for some task and advising on the options offered by a complex computer program.

# S.1 - Technical Review

The following sections present a technical overview of S.1, drawing some examples from the domain of the Framentec "Car Charging Advisor" sample expert system. This system helps an apprentice mechanic diagnose and correct problems with a car's charging system.

## REPRESENTING FACTUAL KNOWLEDGE

Factual knowledge in S.1 is represented as an extension of "object-attribute-value" triples. For example, "The cause of the car's problem is that the fanbelt is slipping", could be represented by identifying "the car" as the central object being discussed, "cause of problem" as the attribute, and "fanbelt slipping" as the value of that attribute. In S.1's language, this set of facts about the world is represented as follows:

**cause.of.problem[CAR] = fanbelt.slipping.**

Objects do not need to be explicitly encoded in advance of a consultation. Rather, they are created as needed by "instantiating" a more general structure, called a "class". As the name implies, a class is a description of a type of object. Individual objects, also called "instances" of classes, are established when they are required.

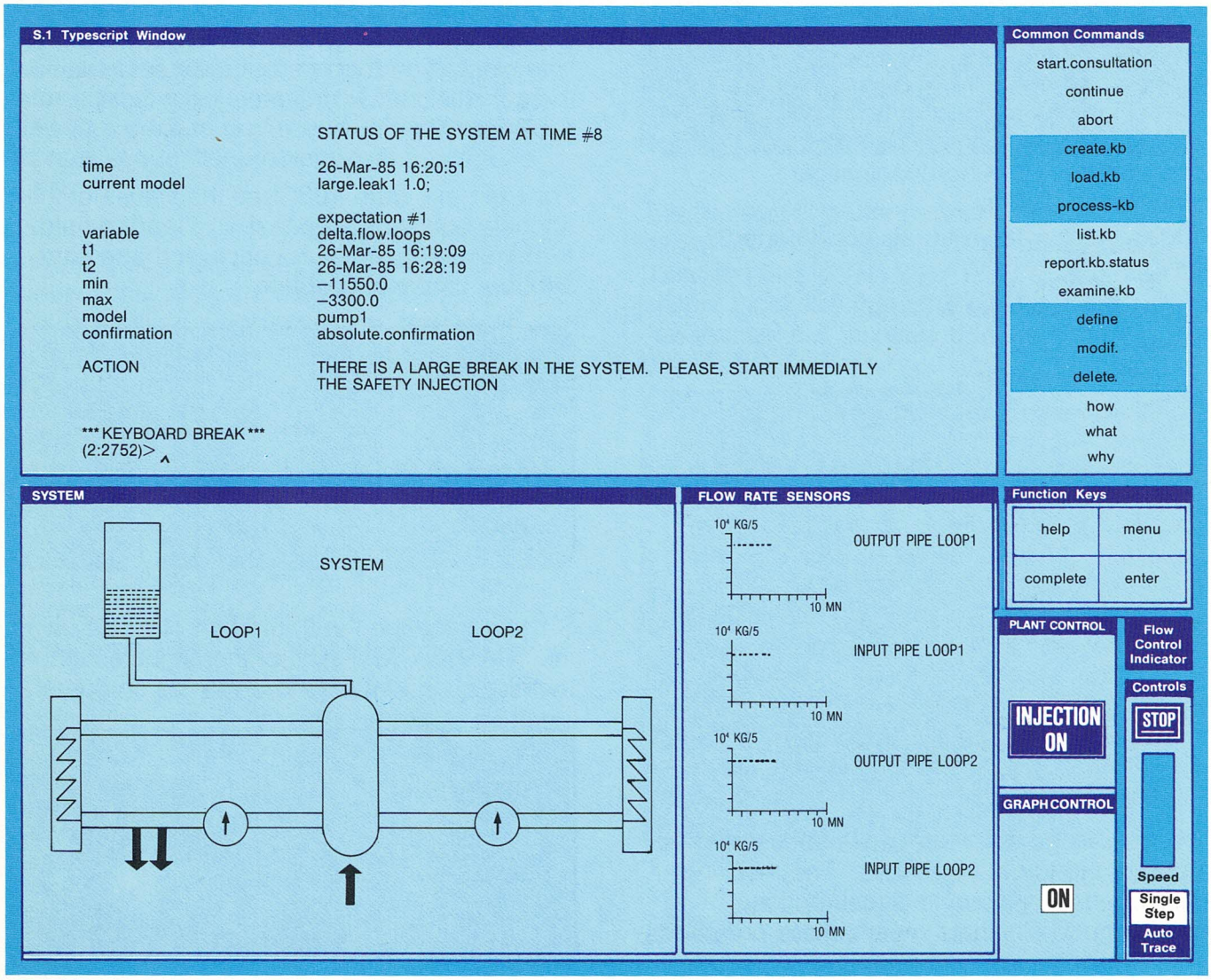
For example, in the Car Charging Advisor system, the focus of a consultation is represented by the class **CAR**:

```
DEFINE CLASS car
   :: NUMBER. INSTANCES                1
   :: PRINT. ID                        "car s"
   :: CLASS. TRANSLATION               "the car"
   :: PLURAL. CLASS. TRANSLATIONS      "the cars"
   :: BLAND. INSTANCE. TRANSLATION     "the car"
   :: FULL. INSTANCE. TRANSLATIONS     "this car"
   :: ANNOUNCEMENT                     new line () ! indent () !
                                       This car will be referred
                                       to as: " !
                                       instance.name (car) !
                                       new.line () ! outdent ()
```

One of the key pieces of information that the Car Charging Advisor needs to determine is the cause of the problem with the car; this is represented by the attribute **cause.of.problem**:

```
DEFINE ATTRIBUTE cause.of.problem
   :: DEFINED.ON                       car
   :: TYPE                             text
   :: MULTIVALUED                      false
   :: LEGAL. VALUES                    hypothesis. values
   :: LEGAL. MEANS                     (try rules)
   :: DETERMINATION.BLOCK              get.cause.of.problem
   :: TRANSLATION                      "the cause of the problem
                                       with the car"
```

**Process control in a Real Time Environment** - Here, the Knowledge Engineer has created new windows to show the occurence of leaks in a power plant.



The possible values for this attribute comprise a hierarchy of problems called **hypothesis.values.** This hierarchy contains such causes as: **battery.dead, fanbelt.slipping,** and **alternator.brushes.worn.out.**

Simple "object-attribute-value" triples are limited in their ability to express complex relationships among objects. Suppose more than one problem can be anticipated for a given car. In S.1, the "problem" can be encoded as another object. The attribute "cause" might then relate both to a problem and to a car, as follows:

**cause[PROBLEM,CAR] = fanbelt.slipping.**

In formal, mathematical terms, an attribute in S.1 is a general mapping from a vector of objects to a set of values. This means that binary functions and relations can be readily represented in S.1.

Attributes can also be related to one another. S.1 includes a useful, common relationship among attributes called "subsumption". Consider the following hypothetical consultation dialog:

) Does the car have an air conditioner?
)) Yes.
) Does the car have a radio?
)) No.
) ... further questions...
) Is the car's radio am-fm or am only?
)) ???

There is no meaningful answer to the last question. One can imagine that the system contains two attributes **has.radio** and **am.fm.radio.** The second attribute is meaningless if the first has "NO" for its value. **Has.radio** can be said to subsume **am.fm.radio.** S.1 allows the knowledge engineer to express the subsumption relation among attributes directly. A complex hierarchy of subsumption relations among attributes can be easily encoded.

The knowledge engineer uses subsumption to organize the knowledge base. The resulting expert system will use its subsumption knowledge to eliminate meaningless or fruitless lines of reasoning.

Classes can also be organized hierarchically, using the "class type" mechanism. Values can similarly be organized into "value hierarchies".

The combination of attribute subsumption, class types, and value hierarchies gives S.1 a specialized frame language for expressing hierarchical relationships and making use of those relationships during consultations.

## REPRESENTING JUDGMENTAL KNOWLEDGE

Not all of the knowledge needed for a task can be represented as facts. Rules are used to represent the judgmental or heuristic knowledge needed to solve a problem. Rules have a very simple form which can be expressed in English as a conditional or **"if- then"** sentence. The **if** part contains an expression involving facts that logicians call a "premise". The **then** part of the rule contains the conclusion. Given some input facts, a rule asserts new facts as true.

For example, **Rule 408** uses the values of the attributes **oscilloscope.test** and **loaded.field. terminal.test** to make a conclusion about the attribute **cause.of.problem :**

```
DEFINE RULE rule 408
   : : APPLIED.TO      c:car
   : : PREMISE         oscilloscope.test (c) is fluctuating.arches
                       and loaded.field.terminal.test (c)
   : : CONCLUSION      cause.of.problem (c) = voltage.reg.bad <.9>
   : : CATEGORIES      (regular)
END.DEFINE
```

S.1 can generate the translation of a rule using the TRANSLATION slots of the attributes that it mentions :

```
rule 408
   C is the car.
   If
      the pattern obtained by attaching an oscilloscope to the charging
      circuit is fluctuating.arches and the alternator does respond
      properly to different loads,
   then
      there is strongly suggestive evidence <.9> that the cause of the
      problem with the car is voltage.reg.bad.
```

The knowledge engineer is able to encode the degree of certainty with which a car mechanic believes that the voltage regulator is bad when fluctuating arches appear on the oscilloscope and the alternator responds properly to different loads. This **certainty factor** reflects the mechanic's level of confidence in the rule. In this case, the mechanic was nearly certain that the problem was a bad voltage regulator. S.1 allows the knowledge engineer to encode certainty factors ranging from –1.0 to 1.0.

## REPRESENTING CONTROL KNOWLEDGE

Along with facts and judgments, there is a third kind of knowledge which characterizes the way humans accomplish tasks; this is control knowledge. Human problem solvers know how to gather relevant information and pursue sub-problems in a productive order. They typically break the task into simple steps or actions which they perform in a familiar sequence. While it is easy to write facts and rules in a declarative fashion, it is difficult to express sequential problem solving steps in this form.

S.1 approaches this problem by introducing a **control block** language that is similar to conventional programming languages. A control block is an object in the knowledge base that contains procedural knowledge expressed in an imperative style familiar to programmers. Control blocks indicate the actions that the expert system should perform, the order in which these actions should be performed, and the conditions under which they should be performed.

A control block can perform the following actions:

- Create an object (instance of a class);
- Determine the value of an attribute for an object, i.e., use all possible means to gather information about the attribute;
- Seek the value of an attribute for an object by querying the user;
- Seek the value on an attribute for an object by applying rules;
- Invoke another control block;
- Display text to the user.

The following control block specifies what actions the Car Charging Advisor performs during a consultation. The Car Charging Advisor discusses a particular car with the end user, finding out first what symptoms were observed with the car, and then determining the cause of the problem and the recommendations to fix the problem. Finally, it displays its recommendations:

```
DEFINE CONTROL.BLOCK about.car
   : : INVOCATION      top.level
   : : TRANSLATION     "diagnose and repair a car"
   : : BODY            begin vars c:car;
                          display spaces(15) ! "Welcome to the Car
                            Charging Diagnosis and Repair Advisor."
                            ! nex.line( ) ;
                          create.instance car called c ;
                          determine symptoms [c] ;
                          invoke initial.checks (c) ;
                          determine systems [c] ;
                          determine cause.of.problem [c] ;
                          determine recommendations [c] ;
                          determine extra.recommendations [c] ;
                          invoke display.recommendations (c) ;
                          determine final.system.check [c] ;
                          invoke display.systems (c)
                       end
END.DEFINE
```

The translation of the control block is generated by the system. It is constructed using the text in TRANSLATION slots of the control block itself and the attributes and classes that are mentioned in the control block's BODY slot:

```
about.car:
   In order to diagnose and repair a car, follow this procedure :
      C is the car
      Display the following : <15 spaces> Welcome to the Car
      Charging Diagnosis and Repair Advisor. <start a new line>
      Find out about the car called C.
      Determine the initial symptoms about the car.
      Do the initial checks on the car.
      Determine the suspected systems of the car.
      Determine the cause of the problem with the car.
      Determine the recommendations for fixing the problem.
      Determine additional recommendations concerning the problem.
      Show the recommendations to fix the problem.
      Determine whether the final system check is done.
      Show the other possible systems that could be causing the
      problem.
```

S.1

# Performing a Consultation

One control block in the knowledge base is designated the "top level" control block. A consultation consists of invoking this control block and performing the sequence of actions it specifies. This control block may in turn invoke rules, examine facts, or execute other control blocks.

In addition to this explicit consultation control, a variety of implicit control methods come into play in the reasoning process.

**Creating an object:**

The definition of a class indicates whether an instance of an object can be created automatically, or whether the user should be queried to see if it exists. Thus, when the system needs to create an object, it may first query the user.

The definition of the class may name a control block that should be invoked whenever an object is created. If so, the system invokes the control block after each object is created.

**Determining an attribute:**

The definition of an attribute specifies either an ordered list of means for determining the attribute's value, or the name of a control block that will do the determination. When the system needs to determine the attribute's value, it interprets the list of means or invokes the control block, as appropriate.

The definition of an attribute may name a control block that should be invoked whenver the attribute is determined for an object. If so, the system invokes this control block after the attribute is determined.

**Applying rules:**

When the system seeks the value of an attribute using rules, it tries only those rules that conclude about that attribute. For example, the system will try **Rule408** to established a value for the attribute **cause.of.problem.**

When a rule is tried, the value of each attribute in its premise is tested. If no value has been determined for some attribute in the premise, the system will attempt to determine a value for that attribute. For example, if the attribute **oscilloscope.test** is not determined before **Rule408** is tried, the system will determine the value of this attribute before testing the condition in the premise of **Rule408.** This is called "goal-directed" determination, to distinguish it from determination that occurs as an action in a control block. If the system applies rules to conclude about an attribute that is determined in this way, the process is called "goal-directed backward chaining of rules".

In examining attributes in the premise, it may turn out that the object for that attribute does not yet exist. Should this occur, an attempt will be made to create that object. This is called "goal-directed" instance creation to distinguish it from creation that occurs as an action in a control block.

Additional inferences may be drawn based on relationships among classes, attributes and values.

When the system creates an object of a class that has a "parent" class, the new object is automatically associated with an object of the parent class. For example, after creating an object **left.front.wheel,** the system could create a link **wheel.of** between this object and the parent object **CAR.**

When it is necessary to find a value for a subsumed attribute (e.g., **am.fm.radio**) the system first checks to be sure that the subsuming attribute (e.g., **has.radio**) has the appropriate value. This may in turn entail determining the value of the subsuming attribute.

If the system happens to conclude a value for a subsumed attribute before a value has been established for a subsuming attribute, (e.g. if a rule concludes that **am.fm.radio** is true before determining **has.radio**), then an appropriate value is concluded for the subsuming attribute as well.

Conversely, if the system concludes a value for a subsuming attribute that makes a subsumed attribute irrelevant, the subsumed attribute is never determined (e.g. if **has.radio** is false, then no value for **am.fm.radio** will ever be sought).

S.1 uses the hierarchical relationships among the values for an attribute to draw additional inferences. For example, if one rule concludes that the cause of the problem is **battery.dead,** another rule that tests whether the problem is a **battery.problem** can succeed, since **battery.dead** is a **battery.problem.** Similarly, if one rule concludes that the cause is not a **wiring.problem,** another rule that tests for a **broken.wire** can fail.

Through the use of a structured representation of facts, judgments and procedures, S.1 provides a powerful, high-level framework for representing and using knowledge. This framework helps the knowledge engineer to build large-scale expert systems with previously unknown performance.

# Hardware and Software Requirements

S.1 operates on Xerox 1100 and 1108* workstations, on DEC VAX** 750/780, and on Symbolics*** machines operation on other Hardware is planned.

* Xerox 1100-series, Xerox 1100 and 1108 are products of Xerox Corporation.

** VAX and VMS are trademarks of Digital Equipment Corporation.

*** Symbolics is a trademark of Symbolics Corporation.

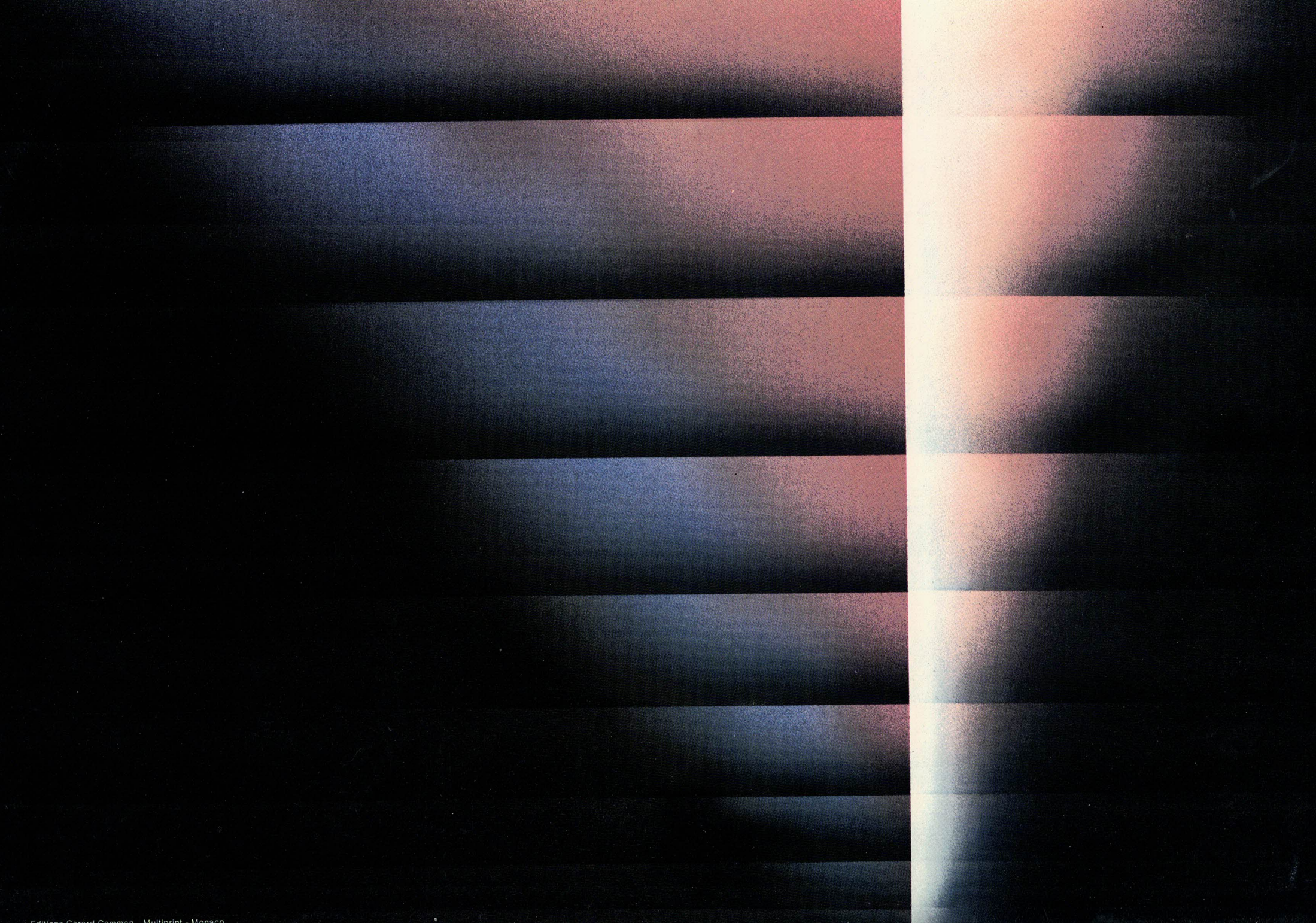
# Course Description

The S.1 course teaches the basic system operation through hands-on laboratory sessions. Instructors provide lectures, guidance, and consultation as participants engage in directed exercises with S.1.

Lecture topics include representing knowledge, designing knowledge bases, creating and reasoning about multiple objects, and representing relationships in S.1.
These topics are illustrated by examining an expert system.

The program is intended for technical personnel who have some experience in building systems using artificial intelligence or knowledge engineering techniques.
Participants should be familiar with basic concepts of representation, inference, and control.

The first week of the course focuses on exercises of increasing complexity and difficulty. The exercises begin with simple additions to existing knowledge bases, leading to the implementation of a small expert system. During the second week, participants perform independent exercises of relevance to their own interests and intended use of S.1. Under the guidance of the training team, questions that arise about the features and use of S.1 are identified and answered. This provides a head start for participants in implementing systems, and establishes a context for transferring development work to the participants' environment.



Editions Gérard Gommon - Multiprint - Monaco