TYPES OF CONTROL STRUCTURE IN EXPERT SYSTEMS

Jean-Pierre Laurent

Département de Mathématiques et Informatique –
Université de Savoie –
Boite Postale 1104-73011 CHAMBERY CEDEX France

INTRODUCTION

The concept of an expert system covers an increasingly large group of software packages which often have more dissimilarities than points in common. We shall not attempt to give a precise definition of an expert system here, because this might impose too restrictive a framework on the rest of our discussion. We shall simply state that, as is generally recognized, an expert system is a piece of software intended to resolve a certain category of problems, that it uses for this purpose a large quantity of knowledge specific to the field in question, and that in each expert system there is a very distinct separation between this knowledge and the procedures which make use of it.

The first stage in constructing an expert system is to consider the elements which will appear in the knowledge base and to choose how to represent and structure them. It is also necessary to choose which control structure, i.e. which operating mechanism (which may also be called an "inference engine") will have the task of manipulating the knowledge base available to it.

Our intention is to consider the use of a knowledge base regardless of the nature of the objects it contains and how they are represented. The selection of objects, their representation, and the method of using the knowledge base are, of course, closely interrelated problems for "real" applications. *It is nevertheless possible to study the various problems raised by use of a knowledge base, relatively independently of the nature and representation of the knowledge.*

This will be our aim throughout the article, and it is within this framework that we shall now specify what we mean by "knowledge base" (KB) and "operating mechanism" (OM).

THE KNOWLEDGE BASE

In computer science literature, a distinction is often made between factual knowledge and deductive knowledge. The relevance of this distinction can easily be seen when dealing with systems such as Mycin [15] which handles triples (objects, attribute, value) and production rules of the type (if conditions then conclusions). Unfortunately, the distinction is not always so clear.

Is a statement such as "iron is a metal" a fact or a rule? It all depends on the context within which it is used. This item of knowledge may be stored like a fact which allows evaluation of the premises of the production rules, but it may very easily be represented by a deduction rule such as "if x is an iron object then x is a metal object". Likewise, it is possible to have a rule enabling the goal "prove that x is metal" to be replaced by the goal "prove that x is iron".

It appears pointless to us to attempt to determine, in the abstract, whether knowledge is factual or deductive. On the other hand, within a physical computer system, the nature of an item of knowledge *is linked to the way it is used and, consequently, to the way it is represented.*

Thus, using the example of a Production Rule system, if the abstract knowledge "iron is a metal" is stored in the form of a triple (iron, nature, metal), it will be considered as factual; if it is stored in the form of a rule (if x is iron, then x is metal), it will be considered as deductive.

It is therefore possible to distinguish between factual knowledge and deductive knowledge at the concrete level of a KB. This will be useful to us when dealing with control problems.

In this paper, we shall consider a knowledge base to be composed of a set of *"objects"*, a set of *"actions"* and a *"halt condition"*.

KB = (Objects + Actions + Halt condition)

Objects

We shall use the term objects for those elements of a KB whose representation is a factual interpretation of the corresponding item of abstract knowledge.

Triples (object, attribute, value), semantic networks and frames are examples of representations frequently used for what we refer to here as objects.

Object types may be very different, depending on the field. The following possible object types shold be noted: HYPOTHESIS, GOAL, SUBGOAL, etc. (between which there may be AND/OR relations). Certainty or probability coefficients may be associated with certain objects, in order to permit approximate reasoning.

Actions

We shall use the term actions for those elements of a KB whose representation is a deductive interpretation of the corresponding item of abstract knowledge.

The most frequent type of representation for actions in current expert systems is the production rule (PR). An action may also be a conventional procedure, a demon, a server, a Horn clause, etc.

Halt condition

In general terms, what we have called the halt condition is a procedure which is triggered by the operating mechanism at the end of each control cycle in order to find out whether the problem in question has been solved.

The system may hold this knowledge in very different forms. It appears convenient to consider it, theoretically at least, as an element of the KB, used during each cycle to ask the questions "HALT CONDITION VERIFIED?".

THE OPERATING MECHANISM

State transitions

When the system is initiated, a certain knowledge base which we shall call the initial state is available. Among the elements of this KB, some may be used to modify the KB itself. In other words, the operating mechanism must product a new KB state by conducting what we call a *state of transition (or transition):*

$$\text{state of KB} \xrightarrow{\text{transition}} \text{new state of KB}$$

If the transition triggered makes a positive response possible in the new state to the question "HALT CONDITION VERIFIED" then the problem raised is, by definition, solved.

The following is therefore a first and simplistic schema for an operating mechanism:

current state ← initial state
repeat
  current state —(transition)→ new current state
until HALT CONDITION

Before proceeding, we should specify the transition concept in greater detail. In our definition, the term transition is applied to the use by the OM of an action in the KB which, when applied to one (or more) elements in the KB, results in a modification to the KB. In the simplest case, the action is applied to an object.

In order to simplify matters, we shall deal only with the simplest case: the generalizations are obvious. We can therefore say that a transition is characterized by:

— the state to which it is applied;
— a pair (object O, action A) of elements of this state;
— the state which it produces.

$$\text{state i of KB} \xrightarrow[\text{(application of A to O)}]{\text{T}} \text{state i + 1 of KB}$$

Therefore, $T = (KB_i, O, A, KB_{i+1})$.

To make things simple we shall write T = (O, A), except where this would lead to ambiguity.

Example. Let object O (triple) be (Bar, Material, Iron) and action A(PR): if Material (x) = Iron, then Nature (x) = Metal. Transition T = (O, A) produces the new object (Bar, Nature, Metal). It is easy to see the distinction between action A and transition T. If there is an object O′ (Pot, Material, Iron), another transition T′ = (O′, A) could produce the object (Pot, Nature, Metal). When an action includes variables (here, x), it may lead to different transitions (different instantiations of the variables).

Search space

In practice, the complexity of the fields in which expert systems are used is such that it is often impossible to restrict oneself to a "straight line" arrangement such as (state 0 → state 1 . . . solution-state). This arrangement would assume that each transition chosen brings us, inevitably, one step closer to a solution. For many problems, and it is well known that expert systems and artificial intelligence are fields marked by a major combinatorial component, a schema of this kind is entirely inadequate. More complex schemes must therefore be provided, since the operating mechanism has to manage the development of what we call the search space. This space is represented in Fig. 1 by a tree whose nodes are states of the KB obtained by the execution of a certain series of transitions.

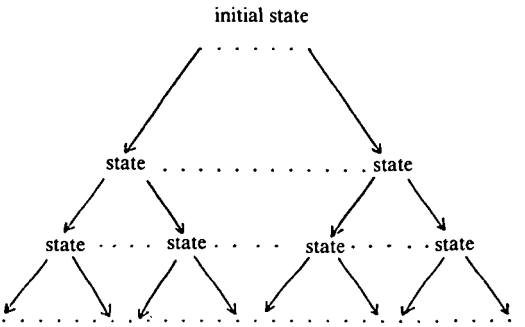

Fig. 1 — Search space

Clearly, the search space may reach a prohibitive size before a solution-state is found. This is usually called the risk of "combinatorial explosion". The greater this risk, the more "intelligently" the OM must manage the development of the search tree, if it is to terminate within a reasonable time and within the size of memory available.

It is this problem of search space management, with a view to solving the problem posed, which we shall deal with in the rest of this article.

Main stages in the control cycle

We shall call the set of operations which enable the OM to produce a new node in the search space, a *control cycle*. In this section, we shall study the various stages in a control cycle.

We shall distinguish between the various possible stages of a control cycle and thus put forward a very general model of an operating mechanism. We shall then see how simplified operating mechanisms can be envisaged, to take account either of a space constraint or of the nature of the problem (which may well make storage of all the memory space unnecessary or even useless).

In general, the following five stages may be distinguished:

stage 1: determination of the set of states capable of being activated
stage 2: state selection
stage 3: determination of the set of possible transitions
stage 4: transition selection
stage 5: execution

State selection. A state which is capable of being activated is any state (any node in the search space) on which all the possible transitions (i.e. transitions enabling a new state to be produced) have not yet been exhausted.

At the beginning, the initial state is the only state which can be activated. When the system has already developed part of the search space, all the nodes (not only the leaves) are states capable of being activated if not all the transition possibilities have been tested.

The first two stages in the control cycle ascertain which states are capable of being activated and then choose one of them.

Transition selection. Assume that the state has been chosen. Listing all possible transitions and selecting one of them are the third and fourth stages in the control cycle. Execution of the transition selected is the fifth stage.

Note 1:
The concepts of forward chaining and backward chaining, which are often used to describe the control mode of an ES, can easily be represented at the level of transition selection; they represent restrictions on the set of transitions to be considered, but not different control structures.

Thus, in forward chaining ("progressive mode" or "data-oriented reasoning") only transitions which involve objects of a particular type, often called "known facts", will be considered.

In backward chaining ("regressive mode" or "goal-oriented reasoning") only transitions which involve objects of the type "goal to be proved" or "hypothesis" will be considered. The role of the transitions will be to replace certain goals with other goals.

In both cases, the role of the OM is to produce transitions of the (state → new state) type. The distinction between forward chaining and backward chaining is not therefore significant in the structure of the control cycle as such. It simply expresses different strategies for the realization of the third stage.

Note 2:
The concept of inference engines with or without variables is also frequently used, particularly for expert systems with production rules. At the level of our present considerations (the different stages of a control cycle), this characteristic is only indirectly significant: the only consequence of the existence of variables in production rules (more generally, in actions) is to increase (often considerably) the cardinality of the set of transitions to be considered, because of the various possible instantiations. Determination of this set (stage 3) is therefore more expensive, and selection (stage 4) more delicate. But this does not alter the structure of the cycle in any way.

STATE SELECTION

In this section, we shall see why the first two stages of the general cycle proposed above are sometimes present and sometimes not, depending on the circumstances. We shall attempt to ascertain the characteristics of the fields for which a state selection is desirable or even essential. We shall then discuss the various possible strategies for dealing with the problem of state selection, whenever it arises.

Fields in which state selection must be considered

Search space, as a tree of states, always exists virtually. We shall not examine whether it is always necessary to produce this tree in practice.

In certain fields, it may be useful or even necessary to allow for the possibility of changing the current state during some cycles and therefore of passing through stages 1 and 2 of the control cycle, whereas in other fields it may be completely useless to consider this possibility and the first two stages will be removed.

We have stressed that a change in the current state is actually a backtrack, with *restoration* of a previous state, while a change of object only directs reasoning differently, without invalidating the effects of previous actions. It is therefore clear that the decision whether or not to consider state selection and thus to retain the first two stages of the general control cycle, depends on *whether or not it is necessary to invalidate previous effects* (or at least some of them). This depends solely on the nature of the transitions; or, more precisely, on the nature of the modifications which a transition effects on a state i to produce the state i + 1.

Two factors must be considered in order to ascertain whether it is necessary to develop an actual search tree (with state storage and backtracking) or whether a simple series of transitions, without a change in state, will suffice. These two factors are: *whether or not transitions can be permuted,* and *whether or not transitions can be reversed.*

There are many fields where transitions are non-permutable, particularly when they represent operations executed in real-time by a robot, and in many other cases. Transitions may be both non-permutable and irreversible. Operations such as drilling a hole or emptying the contents of a receptacle are irreversible. Placing a piece of solder at point A and then placing the end a wire on this point will not produce the same result as the same operations executed in reverse.

One characteristic deserves particular attention: fields in which there are non-permutable operations are those where there are transitions which delete elements from the state to which they are applied. Taking the

word element in its most general sense, this may be deletion of an object, a property, a relation, or a rule, etc. Thus, the action of emptying the contents from a saucepan will involve deletion of the information (SAUCEPAN=FULL); moving the robot from x to y will delete (PLACE(ROBOT)=x) and add (PLACE(ROBOT)=y). The deletion of elements by a transition makes other subsequent transitions impossible. It is therefore necessary to provide a mechanism for restoring previous states.

On the other hand, *one may be tempted to believe that in a field where transitions only add to the information without deleting any of it, restoration of states is unnecessary.* This is not necessarily true, since the additions may have the indirect consequence of preventing transitions. For example, think of the modification of the plausibility of a fact in a Mycin type system: a reduction in plausibility due to the application of a rule may render other rules inapplicable if they include this fact in their premises. *There may therefore be a certain risk in applying the hypothesis outlined above without further consideration,* although it is perfectly applicable in some fields.

To sum up, the problem of state selection may always be posed. If there is no limit on memory, the solution may perhaps be obtained more quickly by way of judicious restorations. It is probably even a necessity in a field where certain transitions are non-permutable and irreversible. On the other hand, in a field where, far from deleting anything, transitions add information which it would be useful to propagate to all the states capable of being activated then it seems sensible to reason on a single state, the "current state". This avoids storing the search space and automatically regulates the propagation problem outlined above. It will still be necessary to be able to select each transition in order for the current state to converge rapidly towards a solution-state.

Implementation of state selection

In this section we shall assume that the characteristics of the domain require the possibility of restoring previous states. Therefore, the first two states of the general cycle appear, at least in certain cycles. In which cycles must the problem of state selection be considered? There are two possible answers: in each cycle or from time to time.

Perpetual reconsideration of the current state. It would seem excessive to consider restoring a previous state and therefore abandoning the current state, at least temporarily, in every cycle. This is true for reasons of efficiency, of course, but also, and perhaps even above all, because it presupposes little confidence in the relevance of the transitions already executed.

Clearly, it is better in general to explore a reasonable distance in one direction before changing direction. This concern with maintaining a certain consistency of reasoning has been outlined in [8] under the *focus of attention principle.*

Occasional reconsideration. State selection in a control cycle will therefore have to be initiated only under specific circumstances.

The obvious case is in dead-ends: during the previous cycle it has been found that no transition was possible on the current state. In this case, backtracking obviously has to take place. The concept of backtracking is not in general associated with dead-ends. For example, one may decide to backtrack regularly, every n cycles. One may also decide to have them only when certain "indicators" light up, i.e. when there are good reasons for doubting the value of continuing on the current state.

Once the backtrack problem has arisen, how should the restart point be chosen. i.e. how should one of the states which are capable of being activated be chosen (this may even be the current state, if we have not reached a dead-end)?

Two types of backtrack may be distinguished: "systematic" backtrack and "selective" backtrack.

Systematic backtrack. This implies the use of an *order relation* on all the states which are capable of being activated. For example, one may use the order in which the various states capable of being activated have been produced and choose the most recent state which can be activated.

This requires a scan direction for the development of the search space. In this specific case, we shall speak of "depth first" search.

Selective backtrack. This is where selection introduces various criteria. Some of these may be general and resemble a systematic backtrack criterion, while others may be specific to certain fields of application or even specific to one particular field of application. We shall give a few examples, moving from the general to the specific:

— giving priority to the last generated (giving priority does not mean obligatorily choosing, which would be a systematic backtrack); this criterion avoids dispersal of the reasoning;
— giving priority to the most recent states (same idea);
— using information obtained during previous cycles: certain states may have been considered during a previous cycle as possible "valuable" restart points (e.g. because during the selection of a transition on this state, a transition other than that chosen appear equally or almost equally valid); it is possible to imagine a stack of privileged restart points, where the stack structure enables the above approach to be taken into account as well;
— dependency directed backtracking; the TMS system [4] decides on a backtrack when it detects a contradiction; examination of this contradiction enables it to select a restart point where the contradiction no longer exists;
— use of the backtrack limitation operator CUT in the case of Prolog;
— use of metarules in PR systems, permitting preselection, from among the states capable of being activated, of those which possess a certain characteristic; these metarules are provided by experts in the field of application and represent metaknowledge of the reasoning methods within the field;
— use of heuristic functions for assessing the distance separating a state capable of being activated and a solution-state. Functions of this kind are often difficult to discover but can be efficient. For example, when theorem proving, the length of expressions, the number of occurrences of a certain symbol, etc. may be considered [14; 9]. This assumes that the concept of a solution-state can be characterized easily (how can the concept of checkmate in chess be usefully formalized to define a distance of this kind?).

Of course, the choice between a systematic backtrack and some more sophisticated backtrack mechanism depends on the field in question. If the search is strongly combinatorial, then a systematic backtrack may be unsuitable. Specific meta-knowledge is therefore necessary to select the restart points more efficiently.

The existence of many criteria for assessing the importance of states makes it necessary both to choose a selective backtrack and to consider the problem as soon as the current state is judged to be unpromising. On the other hand, a systematic backtrack (based on an order relation) and dead-ends will often be associated in fields where there is no effective method of assessing the value of a state.

Figure 2 summarizes the various possibilities outlined in section 3. We shall speak of *S-T-type* (State, Transition) inference engines or *T-type* (Transition) inference engines, depending on whether or not states can be restored.

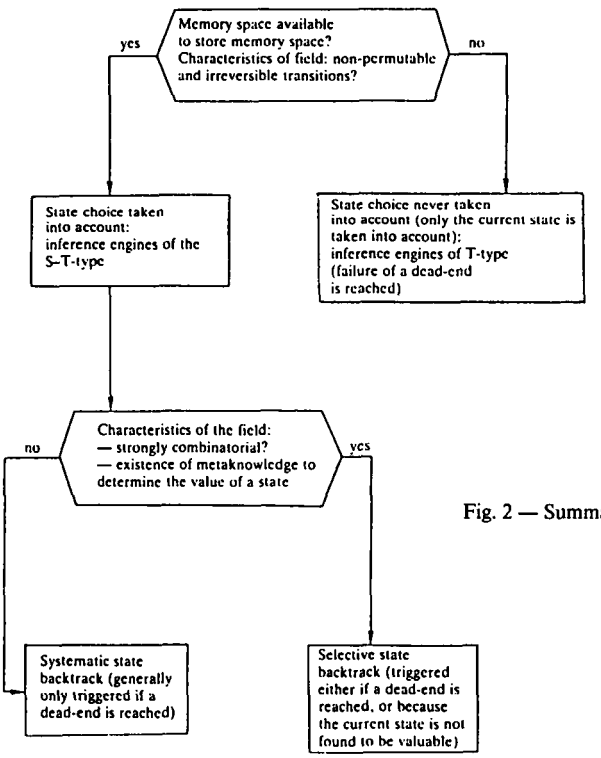

Fig. 2 — Summary of section 3.

TRANSITION SELECTION

We have seen that a transition can be characterized by two elements in the KB: an object and an action. Therefore, to choose a transition on a given state (the current state or the state chosen as the restart point in the event of a backtrack) is to choose one of the possible (Object, Action) pairs.

Throughout this section we shall assume, for the sake of simplicity, that we are dealing with T-type inference engines, for wich there is never a choice of state.

The general cycle is therefore simplified:

stage 1: determination of the set of possible transitions stage 2: transition selection stage 3: execution

We shall now introduce and discuss various possible methods of selecting a transition during each cycle, in other words selecting an (Object, Action) pair. It should, however, be borne in mind that this is a convenient simplification and that, in general, a transition may involve an action and several elements of the KB: one or more object and (or) one or more actions.

Conflict resolution

For many existing engines, e.g. OPS [5] or Snark [12], the transition is selected by choosing one of the (O, A) pairs from the set of possible pairs.

The control arrangement is therefore exactly like that outlined above, but the associated vocabulary is different: the choice of transition is called conflict resolution and the set of possible transitions is called the conflict set:

stage 1: determination of the conflict set stage 2: conflict resolution stage 3: execution

In this case, we shall speak of a C-type (Conflict) inference engine.

Note that the term "conflict set" has been used in different senses by different authors. Our definition is exactly the same as that used in OPS. On the other hand, in Emycin [17] the object resulting in the transition is fixed (this is the current goal-object); so the conflict set denotes the set of actions (in this case, production rules) which are associated with this object and form the set of possible transitions. This regrettable confusion of terms is a good illustration of the problem we shall deal with now: should an (Object, Action) pair be chosen "at one go" or should a distinction be made between object selection and action selection?

With our definition of conflict set, it is clear that conflict resolution simultaneously determines the object and the action which characterize the chosen transition. This procedure has proved to be effective in certain contexts, but is not universally applicable. There is no reason why it should be satisfactory to produce in this way in all cases. On the contrary, we maintain that the criteria relating to object selection and those relating to action selection may be different, making a distinction necessary.

Possible subdivisions of transition selection

We shall now discuss the nature of the criteria which may be involved in choosing a transition. We shall divide them into three categories:

transition criteria, for choosing an (Object, Action) pair at one go
object criteria, for choosing the object independently of the action
action criteria, for choosing the action independently of the object

There is a possible dynamic aspect in transition selection; transitions may be assessed by assessing the objects they produce.

This leads to the formulation of a first rule: *Object criteria may be used as transition criteria (via the objects produced).*

Next, we have to consider the circumstances under which action criteria, in the sense defined above, can actually be applied. What we find is that in certain fields there are criteria which enable the value of an action to be judged. These are criteria which relate to the characteristics of an action, independently of the object to

which they are applied, e.g. the capture move concept in certain games and the simplification rule concept in formal calculus. These criteria are very useful in transition selection. They may be used in the choice of the (Object, Action) pair within the framework of conflict resolution. They may also be used to chose one action or to select a subset of actions before choosing the transition.

We shall now state a second rul: *If, in a given field, object criteria and action criteria exist simultaneously, it may be sensible to divide transition selection into two stages.*

There are thus two possible control structures. The first is the following:

stage 1: determination of objects on which at least one transition is possible
stage 2: object selection (or, more generally, selection of a subset of these objects)
stage 3: determination of the set of actions applicable to the selected object
stage 4: action selection
stage 5: execution of the transition

In this case, we shall refer to *O-A-type* (Object, Action) inference engines.

The second possible type of structure is:

stage 1: determination of actions which can lead to at least one transition
stage 2: action selection (or, more generally, selection of a subset of these actions)
stage 3: determination of the set of objects on which the selected action is applicable
stage 4: object selection
stage 5: execution of the transition

In this case, we shall refer to *A-O-type* (Action, Object) inference engines.

It can be seen that in both cases a preselection takes place before the actual transition is chosen. In the first case the objects are preselected: this is the most common case and the object criteria are usually decisive (e.g. only goal-objects defined in the previous cycle, stack management, are considered, as is the case with Emycin). In the second case, actions are preselected: this occurs less frequently because it is often difficult to exhibit intrinsic action criteria which are considered as having high priority (nevertheless, this is the case with the Wilkins chess program [18].)

It is clearly necessary to examine the characteristics of the field before selecting the type of structure C, O-A, or A-O.

Figure 3 summarizes the various possibilities outlined in sections 4.1 and 4.2 Figure 4 summarizes the possible control cycle architectures, according to distinctions made in section 3 and 4.

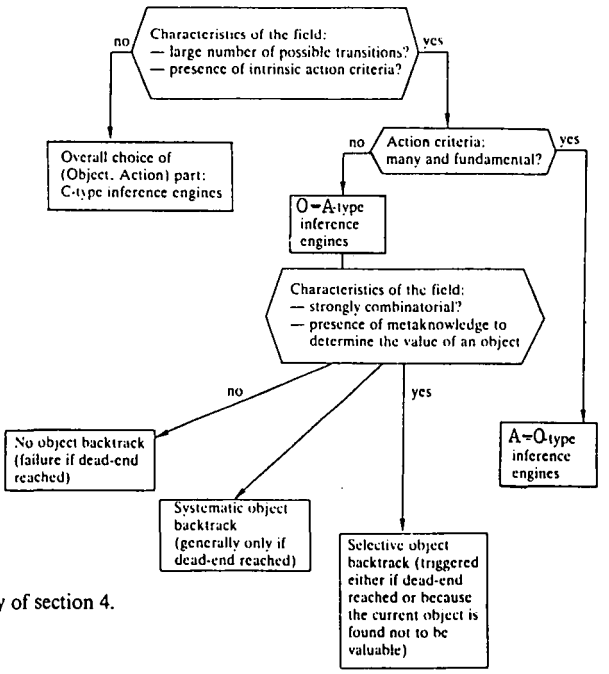

Fig. 3 — Summary of section 4.

Isomorphism between state selection and object selection

If object selection is considered as the first stage in transition selection (O-A structure), all the problems which arose in above for state selection may arise for object selection: anything stated above as regards state selection is therefore valid for object selection.

In particular we can discuss the existence of a "backtrack of object" mechanism instead of a "backtrack of state" one; distinctions between systematic backtrack of object and selective backtrack of objects can also be made.

Note 3: The "isomorphism" between the state-selection-state-backtracking problem and the object-selection-object-backtracking problem is particularly obvious when each state contains a tree of goals deduced from a "root-object" by transitions already applied and when each transition is necessarily applied to a node in this tree (e.g. transitions have the role of producing new goals on the basis of an initial goal: the only difference is that with goals, we are generally concerned with an AND/OR tree, which is not the case with a tree of states).

IMPLEMENTATION PROBLEMS

In the above section, we have discussed various possible stages in a control cycle. We shall outline in this section some methods of implementing these stages. Control in a particular expert system can in fact be characterized by its control cycle structure, but also by the way in which each stage is processed.

Propagation

Some stages in the control cycle consist of determining a set of elements from which choices will later be made. There is an efficiency problem connected with these stages: is it possible to avoid recalculating these various sets for each cycle? Is it possible to deduce them from the corresponding sets for the previous cycle, thus saving time? (this is called propagation).

This question has been examined in particular for expert systems using conflict resolution. Within this framework, given the size of the conflict set, the problem of propagation is particularly important. Forgy has estimated the computing time of the conflict set in the OPS inference engine (a C-type inference engine) at 90% of the total cost of the cycle [5]. This is why he set out to develop a propagation mechanism which would reduce the cost of this stage considerably. This propagation mechanism is characterized by the fact that the indications necessary for updating appear at the end of the action rules. This may seem a fairly severe burden at the rule-writing stage, but it unquestionably permits relatively rapid updating.

Consideration of selection criteria

Once a set of prospective elements has been calculated (or updated by propagation), some stages in the control consist of making a selection from this set. Depending on the fields of application, various types of criteria may be used.

How should a selection stage be implemented, that will use the selection criteria exhibited in the field? Obviously, the answers will vary enormously from one field to another. Nevertheless, we can give a few guidelines on the subject.

Use of order relations. In general, there are orders relating to the storage of elements (of the KB) in memory and chronological production orders (states in the search space, objects generated by previous transitions, etc.). Obviously, the application of criteria of this kind is easy to program, in the conventional procedural sense of the term, in the inference engine.

Use of object criteria and action criteria. The issues here are more complicated.

(a) The consideration of *general criteria* (such as the preselection of the most recent elements produced or the preselection of elements recognized as valuable during previous cycles) may also be *programmed* in the inference engines. The same applies to criteria which are simple, specific and small in number (e.g. a numeric evaluation function for assessing a chessboard situation). Of course, any modification of the selection criteria will have to be followed by program modification.

(b) On the other hand, when there are many criteria to be taken into account, a more flexible solution must be adopted. In the case where there are *many criteria specific* to the field of application, they are liable to be modified very frequently (during the development of the ES and even afterwards during use: the state of the art in the field is, after all, far from fixed). It is also possible for these criteria to represent knowledge (known as metaknowledge) which enables reasoning on actions to take place

(e.g. metarules for reasoning on rules). For all these reasons therefore, it is completely conceivable, and even desirable, that the control structure of the "main" ES refers, during each cycle, to a *"secondary" ES in order to implement a selection stage.* The selection criteria, which permit reasoning on the elements of the main KB (and are usually provided by experts in the field), are added to the latter in order to form the KB of the secondary ES. The role of this secondary ES is to make the selection: its own control structure will have to be defined.

Certainly, the general schema here remains rather theoretical. In practice, the main implementations we are aware of, at this level, are those which, like Teiresias [3] and Centaur [1], use metarules to select a rule. The structure of Teiresias, derived from that of Mycin, is of the O-A type; the object is chosen beforehand and the action is selected using metarules. The latter are expressed in the same form as the rules, and the inference engine of the secondary ES is quite simply that of the main ES. This is therefore a less ambitious implementation than that outlined above. However, the use of a secondary ES (or several) to deal with selection problems arising during each control cycle is, in our opinion, an idea of fundamental importance and an exciting area of research, permitting the design of fairly general, variable engines for many fields of application.

CONCLUSIONS

In this article, we have attempted to define a classification of the various stages which may be included in the control cycle of an expert system inference engine. The distinctions we have made between state selection, conflict resolution, object selection and action selection have enabled us to consider *six possible structures for the control cycle* (Fig. 4).

We have also outlined various types of criteria to enable different choices to be made during each cycle, and we have given several methods of implementing these criteria.

The approach we have taken in pitching our article firmly at the control level, almost ignoring the nature and representation of elements in the KB, may sometimes have appeared rather abstract. It is obvious that the type of knowledge to be handled and the way in which it is represented are important characteristics of any ES, and these characteristics may strongly influence the design of the operating mechanism: *but this influence can be clearly seen in the terms we have proposed. Moreover, we feel that a serious consideration of control should be undertaken before the methods of representation in the base are fixed irreversibly.*

We hope that this paper has achieved the aims laid down: on the one hand, to provide a framework for the comparison of existing systems at operating system level and, on the other, to give designers of new systems a useful framework for their considerations.

A BRIEF DESIGNER'S GUIDE

Let us briefly restate the main general ideas we have put forward:

- the designer should ask himself whether it could be useful, even necessary, to have integral restoration of previous states during reasoning; if so, state selection will have to be included;
- as regards transition selection, object selection and action selection may be either separated or implemented simultaneously through conflict resolution. We have seen that the existence of a large number of "action criteria" may make the first solution preferable;
- if the sets on which selection must be implemented are expensive to recalculate from one cycle to another, propagation mechanisms will have to be considered;
- if a selection stage involves many criteria (in particular metaknowledge specific to the field permitting reasoning on the knowledge in this field), rather than programming selection criteria in the inference engine, it may be preferable to refer during each cycle to a secondary ES which deals with the problem of selection.

These comments should not, of course, be taken as absolute rules but as guidelines which one may attempt to apply to any field. They may or may not be actually applicable.

Overall, two situations may arise:

- after identifying the characteristics which the control structure of the system must have (no doubt after considering the problem of representation at KB level), the designer will be able to choose an operating mechanism with the desired characteristics;
- if a choice of this kind is not possible, the designer will at least have the broad outlines of the operating mechanism that must be produced. One then has only to set to work. . .

REFERENCES

[1] J.S. AIKINS: "Prototypical knowledge for expert systems"; *Artificial Intelligence Journal,* Vol. 20, pp. 163-210, 1983.

[2] B.G. BUCHANAN, R.O. DUDA: *Principles of rule-based Expert Systems;* Stanford University, Dept. of Computer Science, report STAN-CS-82-926, 1982.

[3] R. DAVIS: "META-RULES: Reasoning about control"; *Artificial Intelligence Journal,* Vol. 15, pp. 179-222. Déc. 1980.

[4] J. DOYLE: "A truth maintenance system"; *Artificial Intelligence Journal,* Vol. 12, pp. 231-272, 1979.

[5] C.L. FORGY: *On the efficient implementation of production systems;* Ph.D., Dept. Computer Science, Carnegie-Mellon Univ., Pennsylvania, 1979.

[6] H. GALLAIRE, C. LASSERRE: *Metalevel Control for Logic Programs;* Logic Programming, Academic Press, 1982, 173-185.

[7] I. GOLDSTEIN, R. ROBERTS: "Using frames in scheduling"; *Artificial Intelligence: An M.I.T. perspective,* Vol. 1, pp. 257-284, Ed. P.H. WINSTON, R.H. BROWN, M.I.T. Press 1982.

[8] F. HAYES-ROTH. V.R. LESSER: "Focus of attention in the HEARSAY II Speech understanding system"; *Proc. IJCAI 5,* pp.. 27-35, 1977.

[9] J.P. LAURENT: "DALI, a program that computes limits using heuristics to evaluate the indeterminate forms"; *Artificial Intelligence Journal,* Vol. 4, pp. 69-94, 1973.

[10] J.P. LAURENT, M. AYEL, M. SOUTIF: "CESSOL, un système expert pour définir des campagnes de reconnaissance géotechnique du sol". *4ème Congrès RF-IA de l'AFCET,* tome 2, pp. 393-408. Paris, 1984.

[11] J.P. LAURENT: "La structure de Contrôle dans les Systèmes Experts"; *Revue TSI,* Dunod Ed, Vol. 3 n. 3 1984.

[12] J.L. LAURIERE: "Représentation et utilisation des connaissances"; *T.S.I.,* Vol. 1, n. 1 et 2, 1982.

[13] M.S. LAGRANGE, M. RENAUD; *Simulation d'un raisonnement archéologique, fasc. 1: descriàtion du système SNARK;* Lab. d'Inf. pour les Sc. de l'Homme (C.N.R.S. Paris); 1982.

[14] J. PITRAT: *Un programme de démonstration de théorèmes; monographies AFCET, Dunod ed. 1970.*

[15] E. SHORTLIFFE: *Computer-based medical consultations; MYCIN;* Elsevier, Computer Science Library, 1976.

[16] M. STEFIK, J. AIKINS, R. BALZER, J. BENOIT, L. BIRNBAUM, F. HAYES ROTH, A. SACERDOTI: "The organisation of expert systems, a tutorial"; *Artificial Intelligence Journal,* Vol. 18, pp. 135-173, 1982.

[17] W. VAN MELLE: *A domain-independent system that aids in constructing knowledge-based consutlation programs;* Stanford University, Dept. of Comp. Science, report STAN-CS-80-820, 1980.

[18] D.E. WILKINS: "Using knowledge to control tree searching"; *Artificial Intelligence Journal,* Vol. 18, pp. 1-51, 1982.