COMMISSARIAT A L'ENERGIE ATOMIQUE

CENTRE D'ETUDES NUCLEAIRES DE SACLAY
Service de Documentation
F91191 GIF SUR YVETTE CEDEX

FR 8801379
CEA-CONF --9239

L1

DESCRIBING CONTROL

FOUET J.M.

Centre National de la Recherche Scientifique (CNRS),
94 - Cachan (FR)

STARYNKEVITCH B.

CEA Centre d'Etudes Nucléaires de Saclay, 91 - Gif-sur-Yvette (FR).
Dept. d'Etudes Mécaniques et Thermiques

Communication présentée à : 10. International joint conference on artificial intelligence

Milan (IT)
23-28 Aug 1987

Short paper submitted to 10th IJCAI, Milano, Italy, August 23-28, 1987

DESCRIBING CONTROL

Jean-Marc Fouet (+)
Basile Starynkevitch (++)

(Short paper, Knowledge Representation, Science)
Keywords: Metaknowledge - Control - Heuristics

Abstract

Incremental development and maintenance of large systems imply that control be clearly separated from knowledge. Finding efficient control for a given class of knowledge is itself a matter of expertise, to which knowledge-based methods may and should be applied. We present here two attempts at building root systems that may later be tuned by knowledge engineers, using the semantics of each particular application. These systems are given heuristics in a declarative manner, which they use to control the application of heuristics. Eventually, some heuristics may be used to compile others (or themselves) into efficient pieces of programmed code.

1. Introduction: Control is needed, and cannot be programmed

"Logic must be supplemented by some kind of control structure" [Sim 83]. Not only because it would otherwise be too slow and space consuming, but also in most cases to avoid iterative or recursive explosions. For instance, the rule of example 1 (please refer to the table at the end of this paper) will generate an infinity of facts when applied to any fact. Control is all the more needed if, instead of dealing with pure logic, one considers incomplete and incoherent knowledge bases, rules including programmed predicates (for instance "less than") and actions (for instance "read") with non monotonic side effects (see for instance example 2).

Attempts were made, since Emycin [VMe 80], to draw essential, general purpose inference engines. Algorithms have been proposed [For 82]. But none is really satisfactory: people still write and sell new inference engines, for special purposes, even in propositional logic. In the case of full first order logic, instanciating a variable by a term gives rise to undecidable problems: in other words, any algorithm is a bad algorithm, too heavy for simple cases and too frail for complicated cases.

To be efficient, it seems obvious that control should be event-driven, that it should make the most of properties attached to the objects it handles (constants, predicates and rules), and of its own state (available time and space). This involves semantical information (for instance: "evaluating this predicate costs 10 units") and control heuristics [Len 83]. BUT: (1) a program based on heuristics (a huge number of yet unknown heuristics) will necessarily have to be often modified; (2) giving heuristics in a prescriptive form implies loosing information, in comparison with a possible

+ Laboratoire de Mécanique et Technologie
ENS de Cachan / Univ. ParisVI / CNRS
61 Avenue Wilson - 94230 Cachan - France
tel: (33) (1) 46 64 15 51 Ext 470
++Comissariat à l'Energie Atomique
IRDI/DEDR/DEMT/SERMA/LETR
C.E.N. Saclay - Batiment 70 - 91191 Gif sur Yvette - France
tel: (33) (1) 69 08 40 66

decriptive form (see how many rules can be derived from example 3); and (3), a program cannot easily modify itself. We are therefore developing -each along slightly different principles- EUM [Sta 86] and the Gosseyn Machine [Fou 87] to study the possibility of declarative control; both researches are inspired by the works of D. Lenat and of J. Pitrat [Pit 85].

2. Local control, global control

Control can be seen as the way to answer the question: "what should I do next?". Depending on the level at which the question is asked, possible answers might be: "get a value for that slot", "see if you can match Rule47 with Fact12", or "commit suicide". But control should not only be seen as the way to decide when facing a choice: it also means discovering that there is a choice, planning so as to have no choice, or deciding not to do something even though it was the only possibility.

For each task, two classes of questions arise: why am I doing this (global questions, in relationship with the final goal), and how can I do it (local questions). Every time one meets a tree of possible actions, one has to decide not only which son of a node to consider first (local), but also whether the tree should be developed depth first or breadth first (global). When evaluating a conjunction, the local problem of evaluating each predicate must not hide the global problem: should the predicates be evaluated one at a time, and in which order, or should the whole conjunction be considered. For instance, the first strategy will fail if one tries to find all x's satisfying example 5, since any of the three predicates generates an infinite set. Once it has been globally decided to iterate over a set, local methods should decide whether to iterate from 1 to n, from n to 1, for all even ranks first, etc (see examples 6, 7 and 8).

3. A posteriori control, a priori control

When an item reaches a fork, the decision to send it in one direction rather than the others can be made either by asking for advice or by following a pre-imposed plan. The first method is rather easily implemented: each time a possible choice occurs, the procedure which should (but doesn't want to) choose manufactures a message describing the choice and waits for the reply; the inference engine is called recursively, applying rules (like the one of example 4) to the message. On the other hand, the time needed to match the message with the available rules may soon become so prohibitive that this method cannot reasonably be contemplated, at least on the local levels. Each time knowledge permits, we rather like to spend some time analysing the task before launching it; this results in hints, associated with each item, and aimed at the procedures through which the item should travel. These hints can take a simple form ("when building the conflict set, you should only find two possibilities: discard the one dealing with P..") or can really look like a small program. Of course, when that method fails (the plan cannot be applied), the first one (call for help) is still available.

4. Changing representations

When spying upon the system, one may discover that the same questions are asked over and over again, and the same hints are repetitively manufactured at great expense. It then seems obvious that one should gradually transform knowledge representations so as to diminish the ratio: time spent thinking / time spent acting. Like any other optimization problem, this one involves contradictory criteria. "Caching", for instance, is useful most of the time , but results in an enormous waste of space if used systematically (example 9). Some internal representations are better than others in certain circumstances: we have no arbitrary way of deciding, for instance, between object oriented and relationship oriented paradigms (example 10). For EUM, the ultimate representation consists of C procedures, none of which will have been written by the author, whereas the Gosseyn Machine aims at the design of its own circuits. This very much looks like program specification and control abstraction. In both cases, knowledge is still retained in declarative form, to allow for "conscious" reflection when compiled knowledge fails.

5. Conclusion: bootstrapping

We need good heuristics to manufacture the very good heuristics that will take their place [Pit 85]. We also need a program to use these heuristics until they destroy the program. Writing these programs was not very enthusiasming, but we finally came over it, and are presently tuning the first sets of heuristics. Our representations are highly redundant, including many attributes which are not yet useful. Performances are catastrophic, of course. Once they start improving, we intend to add dynamic introspection (observing one's reactions) to the actual static ability to observe and modify one's structure.

Examples

1. if x does not contain "if-then", then x is not a rule.
2. if x is a task, and x is in the agenda, and x has been satisfied, then remove x from the agenda.
3. if x is a rule, and y is a premise of x, and y is false, then x cannot be fired.
4. if x is a rule, and y was derived from x, then do not apply x to y
5. if x is an integer, and x>5, and x<10...
6. if you are need one an integer, then try 0 and 1 first.
7. if you are looking for elements of a set E, if the cardinality of E is more than 100, try and reduce E first.
8. if you are looking for elements of a set E, if the cardinality of E is infinite, never generate more than one element at a time.
9. if x is a method, if the time spent to find x was more than 10 units, then file x for future needs.
10. if x is a functional predicate, and...., then create a slot for x.

References

For 82 Forgy, C. L., "Rete, a fast algorithm for the Many Patterns/ Many Objects match problem", Artificial Intelligence vol. 19, 1982.

Fou 87 Fouet, J-M., "Compiling Knowledge", submitted to Cognitiva 87, Paris (France), June 1987.

Len 83 Lenat, D. B., "EURISKO", Artificial Intelligence, vol. 21, 1983

Pit 85 Pitrat, J., "MACISTE", Proc. of Colloque Intelligence Artificielle, Toulouse (France), Publications du GR22, 1985.

Sim 83 Simon, H. A., "Search and Reasoning in Problem Solving", Artificial Intelligence, vol. 21, 1983.

Sta 86 Starynkevitch, B., "Toward a self-improving heuristic based A.I. program: the EUM system", Proc. of I.A. Biomed 86, Montpellier (France), September 1986.

VMe 80 Van Melle, W., A domain independant system that aids in constructing knowledge-based consultation programs, Memo HPP-80, Stanford, 1980.