

FORMAL OBJECTS AND FEATURE ASSOCIATIONS IN "ARGOS-II"

Michel CAYROL, Bernard FADE, Henri FARRENY

Laboratoire "Langages et Systèmes Informatiques"
Université Paul Sabatier, 118 Route de Narbonne
31077 Toulouse Cedex FRANCE

ABSTRACT : ARGOS-II is a problem solving system developed at L.S.I. laboratory. Its purpose is to embody a general model of the decision function of a robot. After a brief global characterization of the system, an overview of the initial specialization process is provided. Then two interesting particularities of the system are emphasized : "formal prospective aptitude" and "automatic control of feature associations". ARGOS-II generates action plans but also performs the execution monitoring. ARGOS-II is a pattern directed inference system. The "formal prospective aptitude" allows to continue to build plans even when some circumstances are unknown. They permit to avoid some awkwardnesses or failures and are open to learning. Not only does the "feature associations" mechanism deal with the heavy problem of updating the attributes of a group of objects involved in a unique action, but it also allows to easily create powerful demon functions useful for Artificial Intelligence programming.

1. CHARACTERIZATION OF ARGOS-II [1]

a) ARGOS-II is first a problem-solver : it elaborates the action plans by means of an interpreter of production rules. The rules are invoked by using problem patterns and a pattern-matching procedure. The search method is, fundamentally, of depth-first type in an implicit and/or graph, with back-tracking control facilities. However there is no definite order in the representation of the rules. Their disposition can be modified during the solving process owing to, mainly, the existence of several explicit directives and also efficacious demon families. These directives and demon families are specially designed to deal with rules (a kind of metarules). For example it is possible to inhibit --or to awake-- rules conditionnally or not, temporarily or not. All these possible changes of the rule states are pattern directed. The solving process is not considered as terminated when a plan is produced : ARGOS-II realizes the plan monitoring execution (about the process of monitoring plan execution, see [5],[8],[9]). Besides, the separation of planning process and execution process is not sharp (as it happens rather frequently) : before the whole plan is built, actions can be performed, if it is necessary (deductions, perceptions, interrogations,...). The ARGOS-II program is written in TLISP-language (see [4]).

b) ARGOS-II is also a problem expression language for Artificial Intelligence. It contains many of the primitives implemented in classical procedural languages or in more recent systems which use specifications by production rules

(see [3],[7],[13]. Among the possibilities to act on rules let us point out, not only the above quoted directives to inhibit, to awake or to fire (immediately), but also the facilities to state spontaneous triggering rules and demon rules. All the standard functions of TLISP may be used.

II. OVERVIEW OF THE INITIAL SPECIALIZATION PROCESS

The system maintains several separated associative networks by means of pattern-matching. Essentially "fact-memory", "problem-memory", "rule-memory". For embedding ARGOS-II in a real context, the user initially provides four main kinds of knowledge :

a) facts in like-S-expressions form, e.g. :
(BOX3 ASPECT (METAL BLUE) IN (ROOM1 (5 10)))

b) problem analysis rules in form of production rules :

$$\left\{ \begin{array}{l} \text{rule-name SEX}_0 \text{ SEX}_1 \text{ SEX}_2 \dots \text{SEX}_n \\ \text{problem pattern } (\alpha) \text{ n fact patterns } (\beta) \\ \text{trigger part} \\ \text{====> SEX}'_1 \text{ SEX}'_2 \dots \text{SEX}'_m \\ \text{action part} \end{array} \right\}$$

Example :

Example: $\{ \text{R9 (TAKE } \triangleright X)$
 (α)
 $(\triangleleft X * \text{ IN } \triangleright Y *) \text{ not } (\triangleleft Y * \text{ STATE RADIOACTIVE } *) \}$
 (β)

(γ) (if-patfact (ROBOT * TAKEN <X *) then return)

(δ) (execute (POINT-IN-ROOM (<Y> (>Z)))

(ε) (replace-problem (GO <Z) (APPROACH <X)))

(ζ) (record (plan (GRASP <X>)) (update (modify (ROBOT *) (TAKEN <X>))))}

The words "not", "if-patfact", "...execute", etc... are ARGOS primitives. The symbol "*" represents the operator of indiscriminate absorption (see [14]). We find several important knowledge modules in this rule: utility (α), subsidiary conditions for applicability (β), control depending on facts (γ), communication with environment (interfaces) or knowledge procedures (δ), problem-reduction (in two sub-problems here) (ε), primitive sub-problem (terminal act of the considered robot) and world model updating (ζ).

c) procedures written by means of ARGOS and TLISP primitives. Some of these procedures represent domain knowledge which is useful for internal deductions or computations of the system (e.g.: a method for designing an itinerary, in the case of a moving robot). The others are interfaces between the system and external processes (effectors captors, dialogue with the user). The procedures may be called from the body of the rules by ARGOS-primitives as "execute". In this case, the formal call may contain a list of identifiers for the results: e.g.: (execute (MEASURE-TEMPERATURE (<OBJ <PLACE>) (>X >Y >Z))) becomes after instantiation: (execute (MEASURE-TEMPERATURE (WALL5 UP) (>X >Y >Z))). The real list of results returned as value of MEASURE-TEMPERATURE is then matched with the pattern (>X >Y >Z), thus the results are affected to X,Y,Z.

d) problems in like-S-expressions form, e.g.: (FIND (OBJECT (HEAVY SOLID)))

III. FORMAL PROSPECTIVE APITUDE

In planning phasis, when a production rule is triggered, a part of its arguments receive ground values. Some other arguments may not be instantiated until a posterior step of plan generation or until execution time. ARGOS-II affects an original symbolic identifier to these latter arguments. Defining and dealing with such "formal objects" have already been done by SUSSMAN (see: "wishful thinking technique" in HACKER [11] and by SACERDOTI (see: "use existing objects critic" in NOAH [9]). In ARGOS-II the formal objects are systematically used. The plan generator can produce sequences like:

```
[CHOOSE-FREE (ANVIL)--->(>G005)]
[LOCATE (<G005) ----->(>G006)]
[TAKE (HAMMER) ----->(>G007)]
[BRING-TO (<G007 <G006)]
```

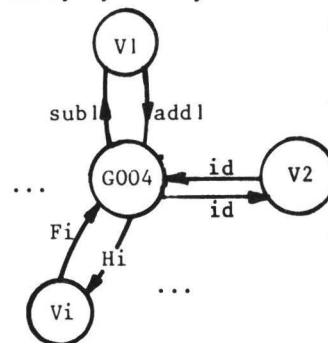
where: "CHOOSE-FREE" and "LOCATE" are planning aid procedures: some occurrences of them did not succeed during planning and entailed the insertion of corresponding steps into the plan to execute; "TAKE" and "BRING-TO" are interfaces with terminal acts of the considered robot; "(ANVIL)" or "(<G007 <G006)" are entry parameter lists; after "→" are described the (symbolic)

expected results of the execution; we note ">X" for a variable "X" which must be instantiated, "<X" the same variable after pattern matching instantiation, "G00i" the original name of the ith encountered formal object.

After the above fourth step some assertions like "(ANVIL <G005 AT <G006)" and "(HAMMER <G007 AT <G006)" will be recorded in fact-memory. These new data may be, in the following of the ARGOS's work, matched with patterns like: "(* ANVIL >X *)" or "(* ANVIL ANVIL4 AT >Y *)". Besides, formal problems may be introduced in the data base and then solved. Such a technique often allows to avoid the awkwardnesses or even failures due to a premature bounding. In spite of the lack of some specifications which can only be obtained at execution time, longer (i.e. "more prospective") plans can be designed. Besides, some impasses which are independent of the unknown parameters can be detected. Let us emphasize that "formal objects" facilitate intermixing planning and execution monitoring, allowing to include perception-decision points in the plans to execute. At last the formal objects process may introduce some kind of learning: a partially symbolic plan, correctly executed may be interesting to save as a resolution scheme (new rule).

IV. AUTOMATIC CONTROL OF "FEATURE ASSOCIATIONS"

The problem of the robot which carries a bag which itself contains a box is well-known: we have to conveniently update some related attributes of a group of objects involved in a unique action. In ARGOS-II is implemented a dispositive which allows to link the transformations of a collection of variables with the modification of anyone among them. The chief primitive available for the user for such a work is "associate": e.g.: (associate (V1 add1 sub1) (V2 id id) ... (Vi Fi Hi) ...) In the above example is defined an association between the variables V1, V2...Vi... via the functions "add1", "sub1", "id", "id", "...Fi", "Hi"... (add1, sub1, id: standard-TLISP-primitives). The association is implemented by linking V1, ...Vi... to a common variable (G004 in the provided example) created dynamically by the system at every use of "associate":



- initially, the value of G004 equals the value of add1 (V1)
- every demand of access to the value of Vi entails an evaluation of Hi(G004) (Hi applied to value of G004)
- reciprocally, every modification of Vi entails an updating of (only) G004, with the value Fi(Vi).

Let us emphasize that for any access or modifi-

cation of Vi only one computation is done, to relate the Vi-value to the common variable value. Indeed, not only explicit variables (as Vi) but also implicit variables (what we call "features") may be handled with "associate". In this case, the formal syntax is of the form : (associate (Pl (Fnl Fi Hl))... (Pi (Fni Fi Hi))...) where Pl...Pi... are patterns, Fnl...Fni... are feature names and Fl, Hl...Fi, Hi... are LISP functions, e.g.: initially, there is in the data base : (PAINT-BRUSH... COLOUR BLUE...) (PAINT-BOX ... COLOUR GREEN...). Let us consider : (associate ((PAINT-BRUSH *) (COLOUR id)) ((PAINT-BOX *) (COLOUR id))). In this example, the first implicitly manipulated feature is the attribute following the prefix "COLOUR" in any datum able to match with the pattern "(PAINT-BRUSH *)". Henceforth the features "COLOUR" of "PAINT-BRUSH" and "PAINT-BOX" will be considered as identical (features "COLOUR", i.e. here, the attributes following the prefixes "COLOUR"). Let us note that, by side effects of the functions Fi and Hi, any procedure can be triggered, either when affecting a value to one of the associated variables or features, or at the moment of access to one of these variables (or features). We can write, for example, very practical procedures to control communications between the robot-system and its environment ; e.g. :

```
(associate (Vl FUNC1 FUNC2))
(define FUNC1 (X) (print PLEASE X ?)(read))
(define FUNC2 (X) (setq HOWMANY (add1 HOWMANY))X)
Any modification of Vl, like "(setq Vl BIG-VALUE)" will modify, in fact, the value of the common variable by call to "FUNC1". The value returned by the computation of FUNC1, i.e. the value of the latter expression "(read)", is affected to the common variable. And for any reference to "Vl", the value computed from the common variable by means of FUNC2 is returned. Besides, "HOWMANY" counts the number of entries by "Vl". The above example shows that, with "associate" primitive family, it is easy to create a new class of demons ; these demons are able not only to watch over the traffic into and out of data bases, but also to watch over the traffic to and from any variable or feature. The facilities provided by "associate" and other analogous primitives implemented in ARGOS-II extend the "AFFIXMENT" statement proposed in AL (see [6],[12]) and the ideas about "representation of actions that have side effects" implemented in AIMDS [10]. They are adapted to current mechanisms used in pattern directed inference systems (e.g.: back-tracking)..
```

V. CONCLUDING REMARKS

ARGOS-II has been designed to be a general system able to control the behaviour of a robot whose particular description is provided by the user. Industrial applications (computer aided production) are being studied. Among several other worth-considering directions of research for the ARGOS project, let us quote : the lear-

ning of plans ; the extension of the system capacities to interact with operator and to explicate the (deep) reasons of its behaviour ; the fuzzification of the triggering mechanism of the production rules (see [2]).

VI. ACKNOWLEDGEMENTS

The authors wish to thank Professor Beaufils, Director of L.S.I. Laboratory, for his constant encouragements.

REFERENCES

- [1] CAYROL M., FADE B., FARRENY H. : Etude en simulation des stratégies de contrôle d'un robot Univ. Paul Sabatier Toulouse Tech. Report Juin 1979
- [2] CAYROL M., FARRENY H., PRADE H. : Fuzzy production rules directed by fuzzy pattern-matching L.S.I. Univ. Paul Sabatier Toulouse Tech. Rpt. Jan. 79
- [3] DAVIS R., BUCHANAN B., SHORTLIFFE E. : Production rules as a representation for a knowledge based consultation program, Artificial Intelligence Vol.8 1977
- [4] DURIEUX J.L. : TLISP Manual Reference L.S.I. Univ. Paul Sabatier Toulouse Juin 1978
- [5] FIKES R.E. : Monitored execution of robot plans produced by STRIPS, Proceedings of the IFIP Congress Ljubljana, Yugoslavia, 1971
- [6] FINKEL R., TAYLOR R., BOLLES R., PAUL R., FELDMAN J. : AL, a programming system for automation, Stanford A.I. Lab. AIM-243 Nov. 1974
- [7] LATOMBE J.C. : Une application de l'Intelligence Artificielle à la conception assistée par ordinateur - TROPIC, Thèse d'Etat Nov. 1977
- [8] NILSSON N.J. : A hierarchical robot planning and execution system SRI A.I. Center Tech. note 76 April 1973
- [9] SACERDOTI E.D. : A structure for plans and behavior, Elsevier Computer Science Lib. 1977
- [10] SRIDHARAN N.S., HAWRUSIK G. : Representation of actions that have side-effects, Proc. 5th I.J.C.A.I. pp.265-266 August 1977
- [11] SUSSMAN G.J. : A computer model of skill acquisition, Elsevier Computer Sci. Lib. 1975
- [12] TAYLOR R.H. : A synthesis of manipulator control programs from task-level specifications, Stanford A.I. Lab. AIM-282
- [13] WATERMAN D.A., HAYES-ROTH F. : Pattern Directed Inference Systems, Academic Press 1978
- [14] WINSTON P.H. : Artificial Intelligence, Addison Wesley Publishing Company 1977