APPROACHES TO KNOWLEDGE ACQUISITION:

THE INSTRUCTABLE PRODUCTION SYSTEM PROJECT

Michael D. Rychener

Carnegie-Mellon University
Department of Computer Science
Schenley Park
Pittsburgh, PA 15213

Abstract

Progress in building systems that acquire knowledge from a variety of sources depends on determining certain functional requirements and ways for them to be met. Experiments have been performed with learning systems having a variety of functional components. The results of these experiments have brought to light deficiencies of various sorts, in systems with various degrees of effectiveness. The components considered here are: interaction language; organization of procedural elements; explanation of system behavior; accommodation to new knowledge; connection of goals with system capabilities; reformulation (mapping) of knowledge; evaluation of behavior; and compilation to achieve efficiency and automaticity. A number of approaches to knowledge acquisition tried within the Instructable Production System (IPS) Project are sketched.*

1. The Instructable Production System Project

The IPS project [6] attempts to build a knowledge acquisition system under a number of constraints. The instructor of the system gains all information about IPS by observing its interactions with its environment (including the instructor). Interaction is to take place in (restricted) natural language. The interaction is mixed initiative, with both participants free to try to influence the direction. Instruction may be about any topic or phenomenon in the system's external or internal environment. Knowledge accumulates over the lifetime of the system.

Throughout these IPS experiments, the underlying knowledge organization has been **Production Systems** (PSs) [2], a form of rule-based system in which learning is formulated as the addition to, and modification of, an unstructured collection of production rules. Behavior is obtained through a simple **recognize-act cycle** with a sophisticated set of principles for resolving conflicts among rules. The dynamic short-term memory of the system is the **Working Memory** (WM), whose contents are matched each cycle to the conditions of rules in the long-term memory, **Production Memory**.

Study of seven major attempts to construct instructable PSs with various orientations leads to recognizing the centrality of eight functional components. Listing the components and their embodiment in various versions of IPS can contribute to research on learning systems in general, by clarifying some of the important subproblems. This discussion is the first overview of the work of the project to date, and indicates its evolutionary development. Members of the IPS project are no longer working together intensively to build an instructable PS, but individual studies that will add to our knowledge about one or more of these components are continuing. Progress on the problem of efficiency of PSs has been important to the IPS project [3], but will not be discussed further here.

2. Essential Functional Components of Instructable Systems

The components listed in this section are to be interpreted loosely as dimensions along which learning systems might vary.

Interaction. The content and form of communications between instructor and IPS can have a lot to do with ease and effectiveness of instruction. In particular, it is important to know how closely communications correspond to internal IPS structures. Similarly, we must ask how well the manifest behavior of IPS indicates its progress on a task. An IPS can have various orientations towards interactions, ranging from passive to active, with maintenance of consistency and assimilation into existing structures.

Organization. Each version of IPS approaches the issue of obtaining coherent behavior by adopting some form of organization of its 'procedural' knowledge. This may involve such techniques as collecting sets of rules into 'methods' and using signal conventions for sequencing. Whether IPS can explain its

*This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

static organization and whether the instructor can see the details of procedural control are important subissues.

Explanation. A key operation in an instructable system is that of explaining how the system has arrived at some behavior, be it correct or incorrect. In the case of wrong behavior, IPS must reveal enough of its processing to allow the more intelligent instructor to determine what knowledge IPS lacks.

Accommodation. Once corrections to IPS's knowledge have been formulated by the instructor, it remains for further interactions with IPS to augment or modify itself. In the IPS framework, these modifications are taken to be changes to the rules of the system, rather than changes to the less permanent WM. As with interaction, IPS can take a passive or active approach to this process.

Connection. Manifest errors are not the only way a system indicates a need for instruction: inability to connect a current problem with existing knowledge that might help in solving it is perhaps a more fundamental one. An IPS needs ways to assimilate problems into an existing knowledge framework, and ways to recognize the applicability of, and discriminate among, existing methods.

Reformulation. Another way that IPS can avoid requiring instruction is for it to reformulate existing knowledge to apply in new circumstances. There are two aspects to this function: finding knowledge that is potentially suitable for mapping, and performing the actual mapping. In contrast to connection, this component involves transformation of knowledge in rules, either permanently or dynamically.

Evaluation. Since the instructor has limited access to what IPS is doing, it is important for IPS to be able to evaluate its own progress, recognizing deficiencies and errors as they occur so that instruction can take place as closely as possible to the dynamic point of error. Defining what progress is and formulating relevant questions to ask to fill gaps in knowledge are two subissues.

Compilation. Rules initially formed as a result of the instructor's input may be amenable to refinements that improve IPS's efficiency. This follows from several factors: during instruction, IPS may be engaged in search or other 'interpretive' execution; instruction may provide IPS with fragments that can only be assembled into efficient form later; and IPS may form rules that are either too general or too specific. Improvement with practice is the psychological analog of this capability. Anderson *et al* [1] have formulated several approaches to compilation.

3. Survey of Approaches

Each attempt to build an IPS has been based on the idea of an initial hand-coded **kernel** system, with enough structure in it to support all further growth by instruction. A kernel establishes the internal representations and the overall approach to instruction. The following are presented in roughly chronological order. Kernel1, ANA, Kernel2 and IPMSL have been fully implemented. The others were suspended at various earlier stages of development, for reasons that were rarely related to substantive or conceptual difficulties.

Kernel Version 1. The starting point for IPS is the adoption of a pure means-ends strategy: given explicit goals, rules are the means to reducing or solving them. Four classes of rules are distinguished: means rules; recognizers of success; recognizers of failure; and evocation of goals from goal-free data. The Kernel1 [6] approach further organizes rules into methods, which group together (via patterns for the same goal) a number of means, tests and failure rules. Interaction consists of language strings that roughly correspond to these methods and to system goals (among which are queries). Keywords in the language give rise to the method sequencing tags and also serve to classify and bound rules. Explanation derives from the piecing together of various goals in WM, along with associated data. The major burden of putting together raw data that may be sufficient for explanation rests on the instructor, a serious weakness.

Additive Successive Approximations (ASA). Some of the drawbacks of Kernel1 can be remedied* by orienting instruction towards fragments of methods that can be more readily refined at later times. Interaction consists of having the instructor point at items in IPS's environment (especially WM) in four ways: condition (for data to be tested), action (for appropriate operators), relevant (for essential data items), and entity (to create a symbol for a new knowledge expression). These designations result in methods that are very loose collections of rules, each of which contributes some small amount towards achievement of the goal. Accommodation is done as **post-modification** of an existing method in its dynamic execution context, through ten method-modification methods.

Analogy. A concerted attempt to deal with issues of connection and accommodation is represented by McDermott's ANA program [4]. ANA starts out with the ability to solve a few very specific problems, and attacks subsequent similar problems by using the methods it has analogically. The starting methods are hand-coded. Connection of a new goal to an existing method takes place via special **method description** rules that are designed to respond to the full class of goals that appear possible for a method to deal with by analogy. An analogy is set up by finding paths through a semantic network containing known objects and actions. As a method operating by analogy executes, rules recognize points where an analogy breaks down. Then general analogy methods are able either to patch the method directly with specific mappings or to query the instructor for new means-ends rules.

*These ideas were introduced by A. Newell in October, 1977.

Problem Spaces. Problem spaces [5]* provide a novel basis for IPS by embedding all behavior and interactions in search. A problem space consists of a collection of knowledge elements that compose states in a space, plus a collection of operators that produce new states from known ones. A problem consists of an initial state, a desired state, and possibly path constraints. Newell's Problem Space Hypothesis (*ibid.*) claims that all goal-oriented cognitive activity occurs in a problem space, not just activity that is sufficiently problematical. Interaction consists of giving IPS problems and search control knowledge (hints as to how to search specific spaces). Every Kernel component must be a problem space too, and thus subject to the same modification processes. The concrete proposal as it now stands concentrates on interaction, explanation (which involves sources of knowledge about the present state of the search), and organization.

Schemas. The use of schemas as a basis for an IPS kernel** make slot-filling the primary information-gathering operation. A slot is implemented as a set of rules. The slots are: executable method; test of completion; assimilation (connects present WM with the schema for a goal); initialization (gathers operands for a method); model (records the instruction episode for later reference); accommodation (records patches to the method); status (records gaps in the knowledge); monitoring (allows careful execution); and organization (records method structure). Orientation towards instruction is active, as in ASA. Explanation consists of interpreting the model slot, and accommodation, of fitting additions into the model. Connection is via a discrimination network composed of the aggregated assimilation slots of all schemas. Compilation is needed here, to map model to method.

Kernel Version 2. An approach with basic ideas similar to ASA and to Waterman's Exemplary Programming [8], Kernel2 [7] focusses on the process of IPS interacting with the instructor to build rules in a dynamic execution context. The instructor essentially steps through the process of achieving a goal, with IPS noting what is done and marking elements for inclusion in the rules to be built when the goal is achieved. Kernel2 includes a semantic network of information about its methods, for use as a 'help' facility. Kernel2 is the basis from which the IPMSL system, below, is built.

Semantic Network. Viewing accumulation of knowledge as additions to a semantic network is the approach taken by the IPMSL system [7]. Interaction consists of definition and modification of nodes in a net, where such nodes are represented completely as rules. Display and net search facilities are provided as aids to explanation and accommodation. The availability of traditional semantic network inferences makes it possible for IPMSL to develop an approach to connection and reformulation, since they provide a set of tools for relating and mapping knowledge into more tractable expressions.

*This approach was formulated by A. Newell and J. Laird in October of 1978.

**Schemas were first proposed for IPS by Rychener, May, 1978

4. Conclusions

The IPS project has not yet succeeded in combining effective versions of components as discussed above, to produce an effective IPS. The components as presently understood and developed, in fact, probably fall short of complete adequacy for such a system. But we have explored and developed a number of approaches to instructability, an exploration that has added to the stock of techniques for exploiting the advantages of PSs. We are encouraged by the ability of the basic PS architecture to enable explorations in a variety of directions and to assume a variety of representations and organizations.

Acknowledgments. Much of the work sketched has been done jointly over the course of several years. Other project members are (in approximate order of duration of commitment to it): Allen Newell, John McDermott, Charles L. Forgy, Kamesh Ramakrishna, Pat Langley, Paul Rosenbloom, and John Laird. Helpful comments on this paper were made by Allen Newell, Jaime Carbonell, David Neves and Robert Akscyn.

References

1. Anderson, J. R., Kline, P. J., and Beasley, C. M. Jr. A Theory of the Acquisition of Cognitive Skills. Tech. Rept. 77-1, Yale University, Dept. of Psychology, January, 1978.

2. Forgy, C. L. OPS4 User's Manual. Tech. Rept. CMU-CS-79-132, Carnegie-Mellon University, Dept. of Computer Science, July, 1979.

3. Forgy, C. L. *On the Efficient Implementation of Production Systems.* Ph.D. Th., Carnegie-Mellon University, Dept. of Computer Science, February 1979.

4. McDermott, J. ANA: An Assimilating and Accommodating Production System. Tech. Rept. CMU-CS-78-156, Carnegie-Mellon University, Dept. of Computer Science, December, 1978. Also appeared in IJCAI-79

5. Newell, A. Reasoning, problem solving and decision processes: the problem space as a fundamental category. In *Attention and Performance VIII*, Nickerson, R., Ed.,Erlbaum, Hillsdale, NJ, 1980.

6. Rychener, M. D. and Newell, A. An instructable production system: basic design issues. In *Pattern-Directed Inference Systems*, Waterman, D. A. and Hayes-Roth, F., Eds., Academic, New York, NY, 1978, pp. 135-153.

7. Rychener, M. D. A Semantic Network of Production Rules in a System for Describing Computer Structures. Tech. Rept. CMU-CS-79-130, Carnegie-Mellon University, Dept. of Computer Science, June, 1979. Also appeared in IJCAI-79

8. Waterman, D. A. Rule-Directed Interactive Transaction Agents: An Approach to Knowledge Acquisition. Tech. Rept. R-2171-ARPA, The Rand Corp., February, 1978.