

Integrating Planning, Learning, and Analogy. A Prototype System

Martin Beckmann (martbeck@uos.de)

Institute of Cognitive Science, University of Osnabrück, Germany

Christopher Lörken (cloerken@uos.de)

Institute of Cognitive Science, University of Osnabrück, Germany

Ute Schmid (schmid@informatik.uni-osnabrueck.de)

Dept. of Mathematics/Computer Science, University of Osnabrück, Germany

Abstract

We present the distributed symbol oriented system IPAL, which is intended to model explorative and inductive strategies of human cognition. IPAL consists of three core modules: A first component explores some finite problem domain by universal planning. A second component extrapolates this knowledge to a general strategy using a folding technique. Alternatively, this induction process can be performed by an analogical reasoning module. We finally discuss a certain model of integration and the opportunities which may arise from its application.

IPAL

IPAL provides a general framework for modeling the acquisition of strategies from problem solving experience: Starting-point is a problem specification, given as primitive operators, their application conditions, and a problem solving goal. Using a planning algorithm, the problem is explored and operator sequences for transforming some initial states into a state fulfilling the

goals are generated. This experience is integrated into a finite program, corresponding roughly to a set of operator chunks or “compiled” knowledge (Anderson, 1998). An important aspect of this finite behavioral program, which makes it different from the cognitive science approaches to operator chunking, is, that objects are not referred to directly by their name but with help of a *selector function*, which is inferred from the structure of the plan. Inference of *selector functions* captures the evolution of perceptual chunks from problem solving experience (Koedinger, 1990). E.g., in Fig.1, the selector function $topof(x)$ is inferred that describes the block lying on top of block x .

Subsequently, this initial experience is extrapolated and generalized to a *recursive program scheme* (RPS) using a *folding* technique (Kitzelmann, Schmid, Mühlpfordt & Wysotzki, 2002). E.g., after exploring a blocksworld problem with three blocks (Fig. 1), a general rule for solving n -block problems is induced. The system infers a domain specific control strategy that simultaneously represents the (goal) structure of the current domain. Alternatively, after initial exploration, a similar problem might be retrieved from memory. That is, the system recognizes that the new problem can be solved with the same strategy as an already known problem - the system performs problem solving by analogy. In this case, a further generalized scheme, representing the abstract strategy for solving both problems is learned via higher order anti-unification. Experience with a blocksworld problem can for example be used to solve numerical problems with the same internal structure. After solving some problems with similar structures, more general schemes evolve and problem solving can be guided by abstract schemes (Schmid, 2001).

Induction of generalized structures from examples is a fundamental characteristic of human intelligence as for example proposed by Chomsky as “language acquisition device” (Chomsky, 1959) or by Holland et al. (1986). This ability to extract general rules from some initial experience is captured in the technique of finite program folding by detecting regularities. The

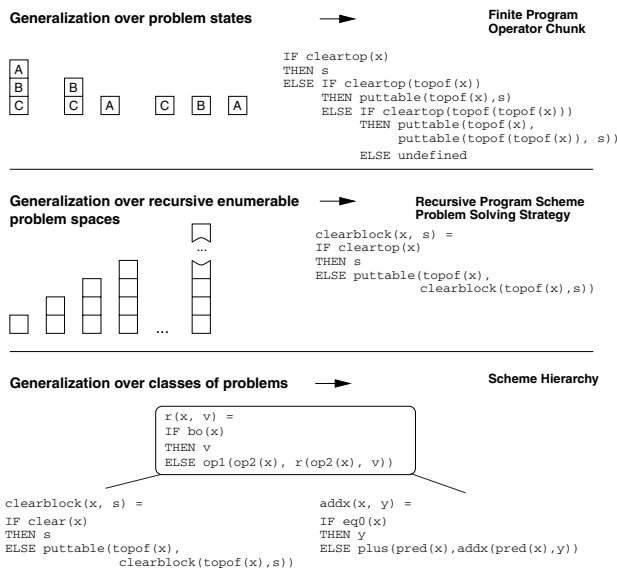


Figure 1: A blocksworld example

representation of problem schemes by recursive program schemes differ from the representation formats proposed in cognitive psychology (Rumelhart, 1981). Nevertheless RPSs are capturing exactly the characteristics, which are attributed to cognitive schemes, namely that schemes represent procedural knowledge (“knowledge how“), which the system can interrogate to produce “knowledge that“ (knowledge about the structure of a problem). Because of this characteristic, problem schemes are suitable for modeling problem solving by analogical transfer (see above) and by abstraction (instantiating an already acquired abstract scheme to solve a new problem).

IPAL in its current form is fragmentary and is open to be extended by further modules for problem solving and learning techniques. E.g., Analogical problem solving could be modeled on the level of plans rather than on finite programs. Following this approach we could enrich traditional planning by machine learning qualities and improve plan construction performance.

To get all the expert units working in a concerted manner and to keep the system extensible it would be useful to integrate it into a standardized interface-based structure. Processing strategies (the inter-modular control flow) should be describable in terms of a sufficiently expressive functional algebra. (E.g., “solve the Tower of Hanoi problem with three discs, induce the generalized n -disc strategy by folding and store it with the domain description”) This language then could become the operation calculus for a *meta-intelligent module* and problems, which can be formulated and solved within it may become subject to machine learning algorithms again.

IPAL Integrated

The first step towards the fusion of a heterogenous set of modules concerns the semantical categorization of the entities involved and the syntactical unification of the *domains* and *co-domains* of all *operators* interacting. (*One type* of knowledge should be represented by *one* syntactical structure.) *Operator* in the above sense refers to the modules again - black boxes from this perspective - which get an input (an element of their *domain*) and then, after some processing return an output (which is an element of their *co-domain*). The intersection of domains and co-domains between the modules obviously defines what are possible control flows through the architecture, or - to speak in terms of the meta calculus postulated before - what is a sound expression. Technically, we use the Hypertext Transfer Protocol for organizing the transmission of symbolic terms from one entity to another. In particular, we are benefiting from the flexibility such a network pattern entails, the interface oriented restrictions imposed do not determine the mode of internal realization, new modules may be implemented in any programming language one could think of. For the future, we intend

that many client structures share a few globally available and increasingly sophisticated servers.

It was said the system obtained its competence to solve more complex analytical and inductive tasks from the controlled interaction of the particular expert units. The server entities themselves are the actively transforming nodes of a network they are completely unaware of. Even a meta intelligence that would be able to “reflect” about the information flow between imaginary nodes (it might also have a representation of itself) has no knowledge about the existential fact that it is actually involved in such a symbiosis. The networks are dynamically assembled on client side.

Hippal

Hippal is the title of our Java based Graphical User Interface, which allows for the composition and instantiation of new functional networks. Hippal is a session oriented system. Whenever a session is created a new logical assembly of modules temporarily comes into existence. We mentioned the possibility of a meta calculus as a means for formalizing module interaction. The concept of a Hippal session is primarily oriented at this calculus. The core component of each session is a central executive, which operates on client objects that themselves communicate with the server entities. From a closer point of view the central executive is an interpreter of a certain syntactical condensation of this meta calculus, a *Lisp dialect* that we have called *LiLi*. A *LiLi* expression is evaluated by performing a well defined sequence of client requests that yields an instance of a particular data type known to the interpreter (e.g. an RPS). Finally, the GUI offers a convenient way to access the interpreter and thus the entire system.

It stays to remark that we do not necessarily claim to imitate human problem solvers in the microstructures of our system but we are modeling human cognition in its macro properties as being a distributed multilayered and multi-strategy based architecture.

Anderson, J. R. and Lebière, C. (1998). The atomic components of thought. Mahwah, NJ: Erlbaum.

Chomsky, N. (1959). Review of Skinner's 'Verbal Behavior'. *Language*, 35, 26-58.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard, P.R. (1986). *Induction - Processes of Inference, Learning, and Discovery*. Cambridge, MA: MIT Press.

Kitzelmann, E., Schmid, U., Mühlpfordt, M., and Wysotzki, F. (2002). Inductive synthesis of functional programs. In J. Calmet et al. (Eds.), *Artificial Intelligence, Automated Reasoning, and Symbolic Computation, Joint International Conference (AISC 2002)*, pp. 26-37, Springer, LNAI 2385.

Koedinger, K. R. and Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry, *Cognitive Science*, 14, 511-550.

Schmid, U. (2001). *Inductive Synthesis of Functional Programs – Learning Domain-Specific Control Rules and Abstract Schemes*. Unpublished Habilitation Thesis (TU Berlin). To be published as Springer LNAI.

<http://www.informatik.uni-osnabrueck.de/schmid/pub-ps/habil.pdf>