AI Magazine Volume 10 Number 4 (1989) (© AAAI)

Penguins Can Make Cake

David Chapman

Until quite recently, it was taken for granted in AI—and cognitive science more broadly—that activity resulted from the creation and execution of plans. In 1985, several researchers, including myself, independently realized that plans and planning are not necessary—or necessarily useful—in activity. Since this time, a number of alternatives have been proposed. In this issue of *AI Magazine*, Matthew Ginsberg, in "Universal Planning: An (Almost) Universally Bad Idea," analyzes one such alternative, Marcel Schoppers's universal plans. He also extends this analysis to a number of other systems, including Pengi (Agre and Chapman 1987), which was designed by Phil Agre and myself.

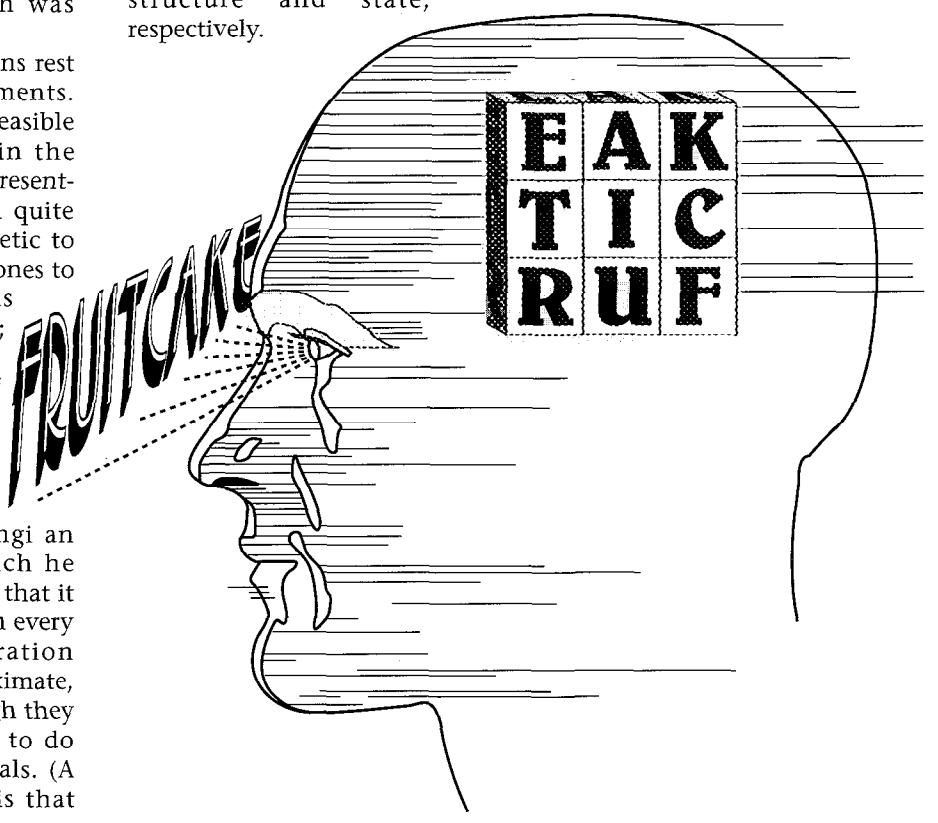
Ginsberg's criticisms of universal plans rest on computational complexity arguments. Using universal plans, he says, is infeasible because their size is exponential in the number of possible domain states. Representing such a plan is infeasible in even quite small realistic domains. I'm sympathetic to such arguments, having made similar ones to the effect that classical planning is infeasible (Agre and Chapman 1988; Chapman 1987b).

I don't understand the details of Schoppers's ideas, so I'm not sure whether this critique of universal plans per se is correct. However, I show that these arguments do not extend to Pengi. Ginsberg calls Pengi an *approximate universal plan,* by which he means it is like a universal plan except that it does not correctly specify what to do in every situation. However, Pengi's operation involves no plans, universal or approximate, and Pengi and universal plans, although they share some motivations, have little to do with each other as technical proposals. (A common related misapprehension is that Pengi is reactive. It isn't.)

The Counting Argument

Ginsberg defines a *universal plan* as a mathematical function from situations to actions and defines *situations* as vectors of instantaneous sensor readings. At the heart of the article is a counting argument, the conclusion of which is that the expected size of a universal plan is exponential in the number of sensors. Presumably, in realistic cases, the number of sensors is large enough that a universal plan could not fit in your head.

There are two reasons not to be concerned about this apparent problem. They involve structure and state, respectively.



0738-4602/89/$3 50 ©1989 AAAI

. . . a Pengi-like system, Blockhead, efficiently solves the fruitcake problem . . .

This article is a reply to Matthew Ginsberg's article entitled "Universal Planning: An (Almost) Universally Bad Idea." Ginsberg argues that universal plans are infeasible for reasons of computational complexity and concludes that classical planning—or something like it—is the appropriate basis for activity. He also argues that a number of other systems, including Pengi, are approximately universal plans and subject to the same criticisms. I think that this extension is incorrect. I illustrate my reasoning with a description of Blockhead, a Pengi-like system that efficiently solves the fruitcake problem which Ginsberg argues is infeasible for universal plans. The structure of Blockhead elucidates the relationship between planning, universal plans, and Pengi. I conclude that planning and universal plans are computationally intractable because of the representational assumptions they make.

Nothing in the counting argument depends on the mapping being from sense vectors to actions. It is really a proof that the size of the smallest circuit to compute an arbitrary function is exponential in the number of its inputs. This analysis is equally applicable to any other computational problem. Thus, you could conclude that vision is impossible because it requires exponential computation in the number of pixels or that, on the average, business data processing takes exponential work in the number of records. However, neither vision nor data processing tasks are arbitrary functions. They have a lot of structure to them, and this structure can be exploited to exponentially reduce the computation's size.

Many theoretically possible inputs are impossible under the rules of the domain, and the remainder can be categorized relatively cheaply to permit abstraction and modularity in their processing. There is every reason to think that this same structure is present in the relationship between perception and activity. Indeed, Ginsberg makes this point himself in arguing for planning: "Our environment behaves in predictable ways; [planning couldn't work if] there were no rhyme or reason to things." Describing and exploiting such regularities is most of the work in designing a system such as Pengi. For further discussion of the sorts of structure found in activity and how it can be exploited, see Agre 1985b, 1988a and 1988b and Agre and Chapman 1988.

Pengi maintains state (in its visual system), so it does not always act the same way given identical sensor readings. This second reason also explains why the counting argument does not apply. Whereas a universal plan is a pure mathematical function from sense vectors to actions (and, hence, does not involve state), state is important to Pengi's operation. John Tsotsos (1989) showed that if vision were computed bottom up (without state) and without constraints on possible inputs, it would indeed take exponential circuitry. However, the addition of a small amount of state can cut this expotential down to a linear function of the number of pixels. Pengi's visual system is quite similar to the one Tsotsos proposes and uses state in just this way.

Blockhead

Ginsberg suggests that the fruitcake problem is not amenable to universal plans and so by implication not to the approach taken in Pengi. In this section, I describe a Pengi-like system, Blockhead, which efficiently solves the fruitcake problem; the way it solves it elucidates what is problematic about both universal plans and planning itself.

The fruitcake problem is to stack a set of labeled blocks so that they spell the word fruitcake. What is apparently difficult about this problem is that the number of situations is factorial in the number of blocks. I show Blockhead solving a problem involving 45 blocks in which there are $45! \approx 10^{56}$ configurations, all of them legal. Blockhead acts correctly in every configuration, so it is not by approximation that it succeeds.

Blockhead has two parts, a visual system and a central system. The power of the architecture lies in the interactions between these components, not in either individually. The visual system is a small subset of Pengi's system, which was inspired by Shimon Ullman's (1984) visual routines theory. The central system is a simple digital circuit (152 gates). The interface between the two systems is intended to be as realistic as possible. The internals of the visual system are not realistic, however; although it conceptually takes pixel arrays as input, it actually has direct access to the blocks world data structures, which made the implementation much easier.

Central to Blockhead's visual system is a limited set of visual markers. *Visual markers* are pointers into the retinal image; they locate points of particular interest. All of Blockhead's access to the image is through these markers; unmarked regions of the image are effectively invisible. The visual system can move the markers around as directed by the central system and can tell the central system about properties of the marked locations. In particular, for each marker, it tells the central

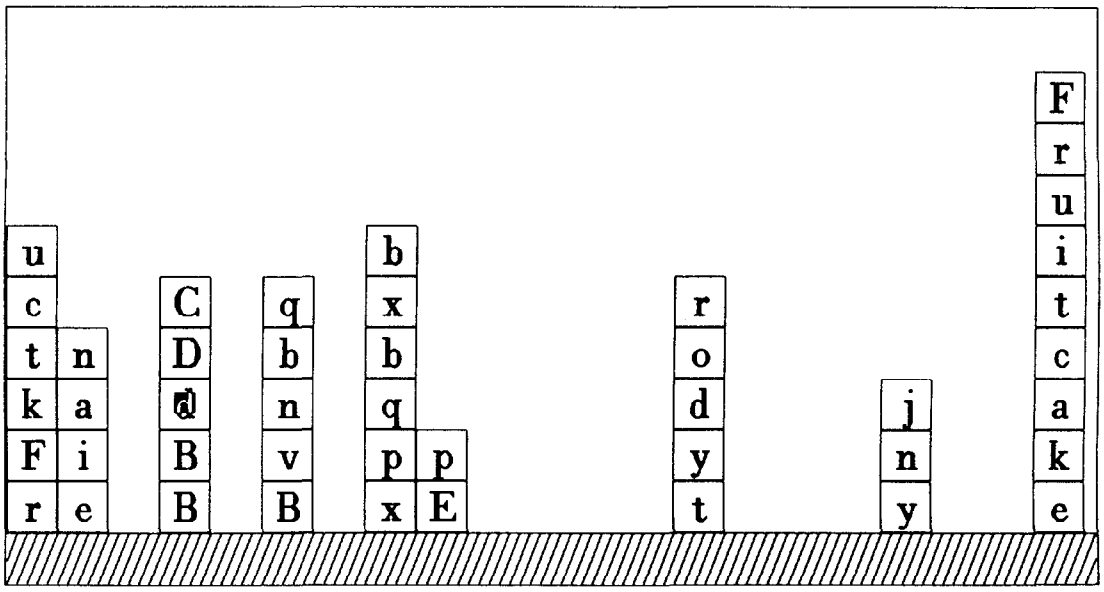


Figure 1. Finding the Bottom of the Stack to Copy in the Initial Situation.

system whether this marker is over a block, the table, or free space. If it is over a block, it can tell what label is on the block. It can also tell whether two markers are coincident; that is, whether they mark the same point. The marker locations are the principal form of state Blockhead maintains.

The visual system provides four operators for moving marks around. It can warp one marker onto another, so that the two are coincident. It can walk a marker a specified distance from its current position in a specified direction. It can jump a marker to the next item in a given direction, ignoring whatever free space is in between. If nothing is in this direction, the visual system informs the central system that the operator has failed. Finally, the visual system can find a point in the image that has specified properties and drop a marker on it. You can specify that the point must be in free space or in the table or in a block; in the last case, you can specify which (alphabetic) label the block must have on it.

With one exception, Blockhead's visual system is domain independent and compatible with available neurophysiological data and can be efficiently implemented on a parallel architecture. There is also considerable psychophysical evidence that Blockhead's visual primitives are available as primitives in the human visual system (Chapman 1989b; Ullman 1984). The exception is the ability to primitively manipulate alphabetic labels. However, colors, which can be primitively manipulated by the human visual system, could be substituted for letters, resulting in a formally isomorphic problem.

Blockhead's effector system consists of only the traditional puton action. The two arguments are specified by means of visual markers.

To make the problem more interesting, I generalized it somewhat: Blockhead will actually copy an arbitrary stack of blocks given to it as a model. Specifically, it copies the rightmost stack that is visible. (This generalization doesn't make the problem any easier; it would be a small modification to the system to solve the problem as stated.)

The first thing that Blockhead needs to do to solve a stack-copying problem is to find the stack to copy. Specifically, it needs to find the bottom of the stack because it will build the copy bottom up. First, it uses the finding operator to put a marker on an arbitrary block. In figure 1, this marker appears as an inverse-video square. In general, the block found might be at any height in a tower; the next objective is to find a block that is on the table. Blockhead accomplishes this task by walking the marker downward, moving a block's height at a time, until the visual system reports that it is into the table (a primitively sensible condition). At this point, Blockhead knows it has gone too far and walks the marker up again, onto the block that is resting on the table. Then, Blockhead repeatedly uses the free-space jumping operator to move the marker right. When jumping fails, the system knows that the square marker is, in

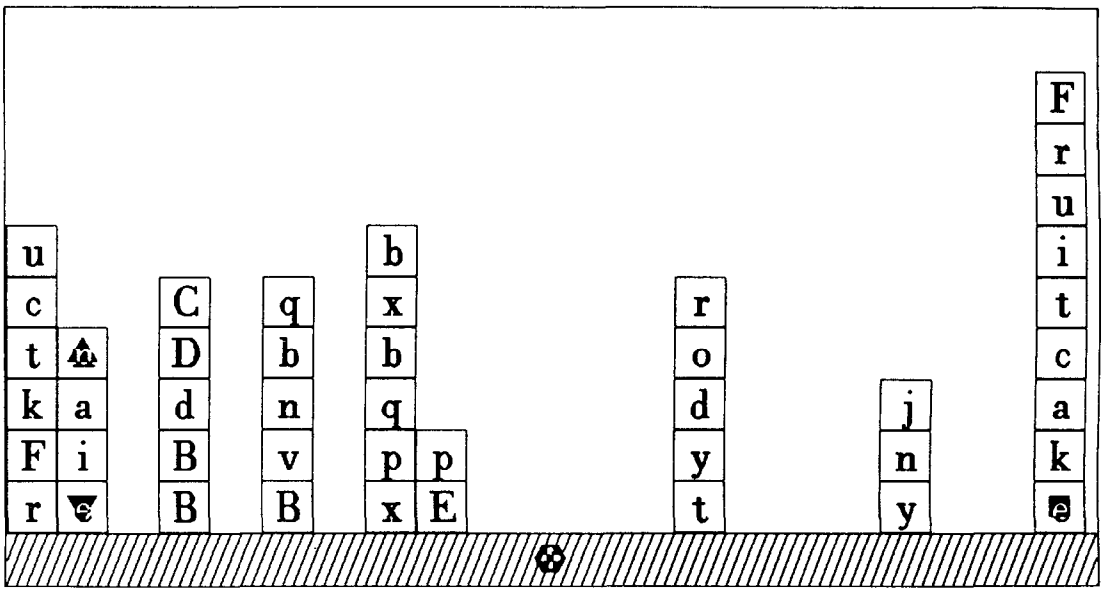


Figure 2. Finding the Top of the Junk on the Desired Block.

fact, on the bottom of the stack to be copied.

The next step is to locate a block that can be the bottom of the new tower. This block has to have the same label as the one that the system has already marked, in this case, e. The system gives this label to the finding operator, thereby finding a block of the desired sort. A second marker, represented by a downward-pointing triangle, is used to label this block (figure 2).

This block might be under a heap of junk that must be cleared away. A third marker, appearing as an upward-pointing triangle, is used to find the top of the stack in which the desired block appears. This marker is first warped to the upward marker, then is repeatedly walked upward until it is found to be hanging in midair. Then it is walked back down onto what the system now knows is the top block.

If the two triangular markers are coincident at this point, Blockhead knows that the desired block has a clear top. Otherwise, it puts the block under the downward marker on the table. It keeps a fourth, cruciform marker on the table at all times; it puts it there using the finding operator right after the system is started up. Thus, giving the puton effector the downward and cruciform markers as arguments removes the junk.

Blockhead then warps the downward marker back to the upward marker in case more than one block was on top of the desired one and repeats this junk-clearing routine until the desired block is found to have a clear top.

Now, the desired block can be put into place. At first, this place is on the table; after this point, it is at the top of the growing copy stack. Another, hexagonal marker is needed to keep track of this stack. Initially, Blockhead puts this marker on the table; as it adds each block to the copy stack, it warps this marker there.

When a block is put in place on the new stack, Blockhead needs to know which is the next block to be copied; so, the square marker is walked up a block's height. If it is over empty space, the system knows it is done (figure 3). Otherwise, it's time to locate another block with the same label, as Blockhead did previously to copy the bottom block. Thus, it repeats the whole single block– copying routine I just described.

Discussion

Why did the fruitcake problem seem difficult for universal plans, and why is it easy for Blockhead? I believe that what makes the problem seem hard is representation and what makes it easy is vision.

Representation is generally thought to make problems easy; if a problem seems hard, you probably need more representation. In concrete activity, however, representation mostly just gets in the way. Virtually every AI approach to the fruitcake problem would start with a database of expressions such as (ON B0648 B1673), (CLEARTOP B0097), and (LABEL-OF B0944 "a"). Representing situations this way differentiates a "skillion" different states that you have to be able to decide what to do with, which leads to a combinatorial explosion. This explosion occurs in classical planning as well as in universal plans; planners spend most of their time creating, elaborating, and deriving properties of models of possible worlds.

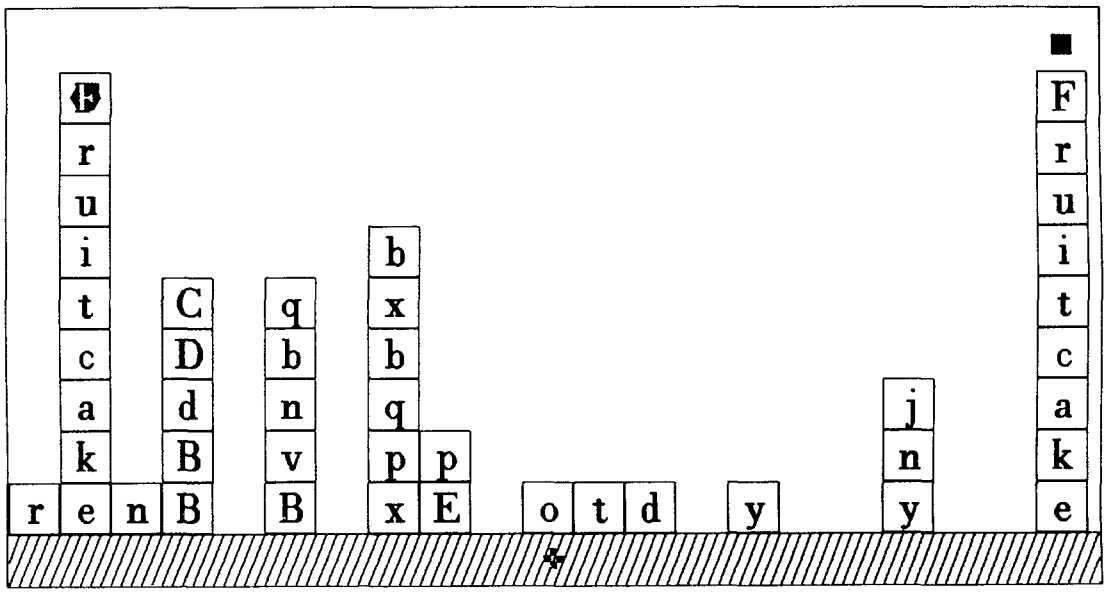


Figure 3. Copying Completed.

Blockhead doesn't have a world model, so it doesn't have to reason about it. It doesn't need one because almost everything in a world model is irrelevant to what the system is up to. For example, if you want to dig a block out from under whatever junk is on top of it, you don't care what the junk is, you just want to dump it all on the table. Avoiding representing irrelevant distinctions collapses the state space. (See Subramanian and Woodfill 1989a, 1989b for formal analysis of this point.) This collapse is a collaboration of the central and visual systems, which together selectively throw away most of the information present in the visual scene. Although any part of the scene can be addressed, only what's currently useful is retained in choosing what to do.

People concerned with action usually think of vision as a hindrance. It doesn't give you the expressional sort of representations you want, and it's unreliable; so, the more you can avoid thinking about it the better. I find that taking vision seriously often actually helps. In this case, applying a serious theory of visual attention tells us that although the visual system supplies as a primitive the ability to track a handful of items using visual markers, representing more items would require considerably more work. Thus, it is clear that you should represent as little as possible, only what's relevant to your purposes.

Of course, Blockhead's visual system is far from serious in general; it ignores most of the interesting issues, such as segmentation and shape matching and noise tolerance and the representation of motion patterns. I think, though, that taking seriously such issues — and analogous issues in effector systems— might make many other apparently difficult problems in activity easy.

Blockhead is a trivial system that I wrote in one day to solve a trivial problem; any architecture in which the fruitcake problem is not trivial would indeed be deficient. Blockhead does not display most of the interesting features of the architecture it shares with Pengi. Pengi is nontrivial because its domain is nontrivial; no planner could survive in it.

I think the best criticisms of Pengi are that it is domain specific, its performance doesn't improve with practice, and it is blind in the face of certain sorts of novel situations. However, these are limitations in Pengi, not in the research program of which it is one product. An ability to acquire new skills from practice might address all three of these criticisms. Agre and I have been centrally concerned with this issue from the beginning (Agre 1985a, 1988a; Chapman and Agre 1987), and in fact, Pengi's architecture is motivated as strongly by concerns of learnability as by those of real-time activity. We are continuing research on skill acquisition (Chapman 1987a, 1989b).

Pengi and Planning

The aim of this article was to show that Ginsberg's critique of universal plans does not apply to Pengi or similar systems. This critique is motivated by a largely implicit comparison between universal plans and classical planning. In reply, I sketch just briefly the relationship between Pengi and planning.

Pengi and planning are not strictly comparable because they have different aims. Planners are designed to solve problems; technically defined, a *problem* is a sort of logical puzzle that can be solved once and for all. Pengi is

. . . entirely new ideas about the nature of plans and their use are needed and . . . these ideas must come from the careful study of human plan use.

designed to lead a life. Pengi's world, like ours, is not a problem to be solved but an ongoing web of recurring opportunities to engage in sorts of activity. Pengi's architecture is good at getting about in the world but not good at solving problems. Planners, being designed to solve problems, are not good at leading lives.

There are many tasks in which plan using and plan making are necessary. It is not yet clear to me, however, how to study them. Since the first wave of nonplanning systems appeared, many researchers have proposed hybrid systems coupling a classical planner with a reactive system. This alternative seems to me to combine the worst features of both approaches. As currently understood, planning is so inherently expensive—and reactive systems so inherently myopic—that even in combination they are useless.

I believe that entirely new ideas about the nature of plans and their use are needed and that these ideas must come from the careful study of human plan use. Preliminary research (Agre and Chapman 1988; Chapman and Agre 1987; Chapman 1989a, 1989b; Agre 1988b) suggests that the relationship between plans and activity is much more subtle and interesting than was previously understood. Plans are not simply executed but are used flexibly and creatively. Much of the work of using a plan is figuring out how to make its text relevant to the concrete situation in front of you. Because plan users are intelligent agents who can be depended on to act sensibly, plans can be drastically simpler than the activity they describe; they need only specify what to do when the right course of action is not clear. Because plans do not have to specify most of what you will do, they can be much easier to make. Because plans are not solutions to problems, the computational intractability of classical planning can be avoided.

Acknowledgments

This article was improved by comments from Phil Agre, Rod Brooks, Mike Dixon, Walter Hamscher, Nils Nilsson, Beth Preston, Jeff Shrager, Penni Sibun, Michael Travers, Dan Weld, and John Woodfill. Thanks also to Matt Ginsberg and Bob Engelmore for giving me this opportunity to reply.

This article describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's AI research is provided in part by the Defense Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124.

References

Agre, P. 1988a. The Dynamic Structure of Everyday Life, Technical Report, MIT AI 1087, Dept. of Computer Science, Massachusetts Institute of Technology.

Agre, P. 1988b. Writing and Representation, MIT AI WP 315, Dept. of Computer Science, Massachusetts Institute of Technology.

Agre, P. 1985a Routines, MIT AI Memo 828, Dept. of Computer Science, Massachusetts Institute of Technology

Agre, P. 1985b. The Structures of Everyday Life, MIT AI WP 267, Dept. of Computer Science, Massachusetts Institute of Technology.

Agre, P., and Chapman, D. 1988. What Are Plans For? MIT AI Memo 1050, Dept. of Computer Science, Massachusetts Institute of Technology. Revised version to appear in 1990. *New Architectures for Autonomous Agents: Task-Level Decomposition and Emergent Functionality,* ed. Pattie Maes. Cambridge, Mass.: MIT Press. Forthcoming.

Agre, P., and Chapman, D. 1987. Pengi: An Implementation of a Theory of Activity. In Proceedings of the Sixth National Conference on Artificial Intelligence, 268–272. Menlo Park, Calif: American Association for Artificial Intelligence.

Chapman, D. 1989a. From Planning to Instruction Use. In Proceedings of the Rochester Planning Workshop: From Formal Systems to Practical Systems, 100–104. Rochester, N.Y.: University of Rochester Department of Computer Science

Chapman, D. 1989b. Instruction Use in Situated Activity. Ph.D. diss., Dept. of Computer Science, Massachusetts Institute of Technology. Forthcoming.

Chapman, D. 1987a. Articulation and Experience. Cambridge, Mass.: MIT.

Chapman, D. 1987b. Planning for Conjunctive Goals. *Artificial Intelligence* 32:333–377.

Chapman, D., and Agre, P. 1987. Abstract Reasoning as Emergent from Concrete Activity. In *Reasoning about Actions and Plans,* eds. M. Georgeff and A. Lansky, 411–424. San Mateo, Calif.: Morgan Kaufmann.

Subramanian, D., and Woodfill, J. 1989a. Making Situation Calculus Indexical. In Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, 467–474. San Mateo, Calif.: Morgan Kaufmann.

Subramanian, D., and Woodfill, J. 1989b. Subjective Ontologies. In Proceedings of the American Association for Artificial Intelligence Spring Symposium on Limited Rationality. Menlo Park, Calif.: American Association for Artificial Intelligence. Forthcoming.

Tsotsos, J. 1989. The Complexity of Perceptual Search Tasks, Technical Report, RBCV-TR-89-28, Dept. of Computer Science, Univ. of Toronto.

Ullman, S. 1984. Visual Routines. *Cognition* 18:97–159.

David Chapman is a graduate student at the MIT Artificial Intelligence Laboratory.