Technical Report 1085

The Dynamic Structure of Everyday Life

Philip E. Agre

MIT Artificial Intelligence Laboratory

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR 1085	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Dynamic Structures of Everyday Life		5. TYPE OF REPORT & PERIOD COVERED technical report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip E. Agre		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE October 1988 13. NUMBER OF PAGES 282
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

situated activity
planning
action
deictic representation
visual routines

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Computational theories of action have generally understood the organized nature of human activity in terms of the construction and execution of computer-program-like structures called plans. By consigning the phenomena of contingency and improvisation to peripheral roles, this view of activity has led to grossly impractical technical proposals. I would like to propose an alternative view of human activity. According to this view, contingency is a central feature of the world of everyday activity and improvisation is the principal means by which people get along in the world. Starting from these premises, I (over)

offer a computational model of certain aspects of everyday routine activity. This model is based on two ideas, a way of organizing improvised activity called *running arguments* and an account of representation for situated agents called *deictic representation.*

A running argument means continually redeciding what to do. Continually redeciding what to do is more flexible than executing a plan because it is more responsive to opportunities and contingencies. It is possible to approximate the ideal of continual redecision because life is almost wholly routine. The routine portion of the reasoning leading to each moment's action can be implemented very efficiently by recording the reasons behind any novel bits of reasoning, a method known as dependency maintenance. A computer program called the running argument system illustrates this point.

Deictic representation means individuating things in the world not objectively (independently of the agent's location or heading or projects or attitudes) but rather indexically (in terms of their relation to the agent) and functionally (in terms of the role they play in the agent's ongoing projects). Deictic representation does not involve a notion of objective identity, but then objective identity is rarely a help, usually a hindrance, and always much too great an epistemic problem to make into a central representational category. A computer program called Pengi illustrates the use of deictic representation.

The Dynamic Structure of Everyday Life

by

Philip E. Agre

Submitted to
the Department of Electrical Engineering and Computer Science
on October 12, 1988
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

Computational theories of action have generally understood the organized nature of human activity in terms of the construction and execution of computer-program-like structures called plans. By consigning the phenomena of contingency and improvisation to peripheral roles, this view of activity has led to grossly impractical technical proposals. I would like to propose an alternative view of human activity. According to this view, contingency is a central feature of the world of everyday activity and improvisation is the principal means by which people get along in the world. Starting from these premises, I offer a computational model of certain aspects of everyday routine activity. This model is based on two ideas, a way of organizing improvised activity called *running arguments* and an account of representation for situated agents called *deictic representation.*

A running argument means continually redeciding what to do. Continually redeciding what to do is more flexible than executing a plan because it is more responsive to opportunities and contingencies. It is possible to approximate the ideal of continual redecision because life is almost wholly routine. The routine portion of the reasoning leading to each moment's action can be implemented very efficiently by recording the reasons behind any novel bits of reasoning, a method known as dependency maintenance. A computer program called the running argument system illustrates this point.

Deictic representation means individuating things in the world not objectively (independently of the agent's location or heading or projects or attitudes) but rather indexically (in terms of their relation to the agent) and functionally (in terms of the role they play in the agent's ongoing projects). Deictic representation does not involve a notion of objective identity, but then objective identity is rarely a help, usually a hindrance, and always much too great an epistemic problem to make into a central representational category. A computer program called Pengi illustrates the use of deictic representation.

Thesis Supervisor: J. Michael Brady
Professor of Information Engineering, University of Oxford

David Chapman spent a million hours listening to all the groping intermediate forms of my ideas. Over several years of intellectual coevolution, we never grew too close together or too far apart. And when I was despairing at my overly ambitious attempts to implement my second round of ideas, he saved my life by writing Pengi.

Mike Brady supervised my thesis work. Mike has an extraordinary understanding of the connection between a long-term vision and the day-to-day of getting things done. His firm sense of what is important and what isn't helped make a long and harrowing process intelligible.

An awful lot of people provided helpful comments, suggestions, and arguments. Discussions with John Batali, Jeff Shrager, and Randy Trigg were particularly important in the development of my thinking. Lucy Suchman introduced me to the intellectual cultures within which I first believed what I heard myself saying.

Aside from the aforementioned, Jonathan Amsterdam, Rod Brooks, Mike Dixon, Carl Feynman, and Eric Saund wrote helpful comments on drafts. Rod Brooks was also the stateside representative of my thesis committee. Patrick Winston shared his experience and served as my third reader.

The Fannie and John Hertz Foundation paid for the first five and a half years of my graduate work.

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

Contents

A Studying everyday life 7

A1 Context and summary 9
- A1a Activity . . . 9
- A1b What I did . . . 12
- A1c Everyday life . . . 13
- A1d Technical and narrative language . . . 16

A2 Machinery and dynamics 17
- A2a Context and summary . . . 17
- A2b Mentalism and interactionism . . . 19
- A2c Machinery and dynamics . . . 22
- A2d Why be concerned with dynamics? . . . 28
- A2e Architecture and personality . . . 30
- A2f Complexity and efficiency . . . 31
- A2g Interactionism abroad . . . 34

A3 Walking to the subway 39
- A3a Context and summary . . . 39
- A3b The route . . . 42
- A3c Edinboro St . . . 44
- A3d Crossing Essex St . . . 46
- A3e L'Avenue de Lafayette and the parking lot . . . 47
- A3f Chauncy St and the subway entrance . . . 49
- A3g About stories . . . 50

B Running arguments 53

B1 Context and summary 55

B2 Dependency maintenance 59
- B2a Context and summary . . . 59
- B2b Main ideas of dependency maintenance . . . 60

B2c Dependency networks 69
B2d Dependencies as a simple account of many things 71
B2e About routines 77
B2f Objections 81

B3 The initial implementation **84**
B3a Context and summary 84
B3b Organization 85
B3c Example 87
B3d Rule language semantics 88
B3e How it works 93
B3f How it works (details) 99
B3g Incremental updating 100
B3h Advanced rule-writing 102

B4 Running arguments **105**
B4a Context and summary 105
B4b Planning vs knowing what you're doing 106
B4c Hierarchy as datastructure vs dynamic 114
B4d Argument and centralization 125
B4e Running arguments 132

B5 Experiments with running arguments **136**
B5a Introduction 136
B5b Architecture 137
B5c Demonstration 140
B5d Patterns of transfer 147
B5e Transfer and goal structure 154
B5f Conclusion 164

C Deictic representation **169**

C1 Context and summary **171**

C2 On connectionism **175**
C2a Context and summary 175
C2b Connectionism reminds us of what's important 176
C2c Beyond 'control' 179
C2d Beyond datastructures 182
C2e Beyond variables and constants 186

C3 Pengi **189**
C3a Context and summary 189
C3b Deictic representation 190
C3c Pengo and Pengi 195
C3d Entities and aspects in Pengi 198
C3e How Pengi decides what to do 199
C3f Seriality and focus 203
C3g Objections 205

C4 How Pengi works **212**
C4a Context and summary 212
C4b Architecture 212
C4c Visual system 217
C4d Central system 224
C4e Example 227

C5 Related work **234**
C5a Context and summary 234
C5b The classical Planning literature 235
C5c Extended Planning schemes 237
C5d Production systems, ACT*, and Soar 240
C5e Situated automata theory 244
C5f The MIT mobile robot group 247

C6 Thinking about the background **250**
C6a Context and summary 250
C6b Three stories about the background 251
C6c First story: The Oxford robotics lab 252
C6d Second story: The electric kettle in my Oxford office 255
C6e Third story: Moving into my Oxford office 259
C6f The background: Phenomenology and dynamics 264

Part A

Studying everyday life

Chapter A1

Context and summary

A1a Activity

Everyday life is almost wholly routine, an intricate dance between someone who is trying to get something done and a fundamentally benign world that is continually shaped by the bustle of human activity. I have been exploring the connections between the organization of everyday routine activity and the organization of the human cognitive architecture.

The project of relating a theory of activity to a theory of machinery offers great promise and raises great difficulties. The promise is that a serious theory of everyday activity as a whole can provide a firm basis for the engineering analyses required by serious computational research, whether this research is intended to produce an explanatory theory of human beings or a justified design for an autonomous robot.

The difficulty is that our existing technical vocabulary is not well suited for designing and analyzing devices that continually interact with their environments. While this conventional vocabulary has been helpful in getting computer technology off the ground, its presuppositions and shortcomings now confuse and disrupt efforts to build systems that can interact flexibly with real environments. The beginnings of a principled alternative lie in careful attention to the nature of routine interactions between sensible agents and benign worlds.

The job of a theory of activity is to describe and explain the ordinary everyday activities of ordinary people. Investigations of activity begin with the idea that human activity has an internal logic of its own. While the organization of human activity is deeply affected by its history, it also has an inherent orderliness, coherence, and laws of change. In short, human activity is the way it is for a good reason.

A great deal is known about the nature of human activity. Perhaps the principal contribution of twentieth century sociology has been to convincingly relate broad questions of social order to the finest details of everyday activity. Among the foundational works in this tradition were careful descriptions of the structure of everyday experience: Heidegger's account of everyday routine activities (1927) and Schutz's account of

everyday social interactions (1932). More recently, profound insights into the social organization of routine activities have arisen from investigations of everyday cognition (see for example Bourdieu 1977, Garfinkel 1967, Lave 1988, Rogoff and Lave 1984, Scribner 1984, Wertsch 1985). Much of this work is marvelous for its observational acuity and its theoretical rigor.

The theme of connecting large-scale phenomena to the detailed organization of everyday activity is also central to my own work. For clarity, let us distinguish a particular 'action' from 'activity' in the large. A theory of action explains how individuals come to do particular things in particular situations. The notion of activity, by contrast, relates to the broader organization of human doings; a theory of activity should describe how individuals' actions are knitted into larger structures of relationships and societies.

My own interest in theories of activity began with my dissatisfaction with the ideas about action I was taught in classes on artificial intelligence. These ideas center on the notion of 'planning'. Perhaps the most influential early statement of the idea of 'planning' occurs in a book called *Plans and the Structure of Behavior*, published in 1960 by Miller, Galanter, and Pribram. These authors rejected the extreme behaviorist view that the organized nature of activity results from isolated responses to isolated stimuli. Instead, they adopted the opposite extreme view that the organization of human activity results entirely from the execution of Plans.

What is a Plan? "A Plan is any hierarchical process in the organism that can control the order in which a sequence of operations is to be performed" (page 16). They state, as a "scientific hypothesis" about which they are "reasonably confident," that a Plan is "essentially the same as a program for a computer," a connotation the term has carried to the present day. Shortly thereafter, though, they state that "we shall also use the term 'Plan' to designate a rough sketch of some course of action, just the major topic headings in the outline, as well as the completely detailed specification of every detailed operation" (page 17). Thus a new Plan's hierarchical structure need not initially reach down to the most primitive actions, though the hierarchy must be filled out by the time any given step of it is executed. This view of Plans and their role in organizing activity has been so long unquestioned within AI that, as one author pointedly observed in summarizing a recent workshop discussion, "the alternatives are not yet clear" (Linden 1988, page 121).

(Unlike subsequent authors, Miller, Galanter, and Pribram were careful to capitalize their version of 'Plan' to avoid confusion with vernacular usage. For them, capitalizing the word emphasized the hierarchical nature of Plans. I will capitalize the word for clarity as well, but in order to emphasize the resemblance between Plans and computer programs.)

Miller, Galanter, and Pribram worked from a distinction and a policy that have remained almost axiomatic in artificial intelligence research down to the present day. They distinguished two activities, 'Planning' and 'execution'. (I will capitalize 'Planning' as well for consistency even though they did not. They imagined that most Plans are retrieved from a collection of stored Plans and had little to say about the actual

process of formulating new Plans.) Beginning with a 'goal', one chooses an appropriate 'Plan' and then one 'executes' it. They define execution by saying that "a creature is executing a particular Plan when in fact that Plan is controlling the sequence of operations he is carrying out" (page 17).

Their policy was to apply the term 'Plan' as broadly as they possibly could. Chapter by chapter, they marched through the various aspects of everyday life, focusing on elements of intentionality, regularity, and goal-directedness and applying the term 'Plan' to each one. One of the many frustrations inherent in trying to argue against Planning is that absolutely any phenomenon in human life can, if looked at right, be viewed as an instance of it. In order to view actions as instances of Planning, though, important aspects of activity must be consigned to peripheral vision. And anybody can concoct convenient toy 'problems' that sound like bits of human activity. But principled research into computational models of action must make explicit its views about the nature of activity as a whole.

The notions of 'Plans' and 'Planning' that descend from Miller, Galanter, and Pribram carry with them a definite view of everyday activity. Subsequent researchers have proposed a variety of technical specifications for 'Planning', 'execution', *et al.* But all of them, whether explicitly or through their continued use of the terms, have shared this view.

> If an agent's activity has a certain organization, that is solely because the agent constructs and deploys a symbolic representation of that activity, namely a plan.
>
> Everyday activity is fundamentally planned; contingency is a marginal phenomenon. An agent conducts its everyday activity entirely by constructing and deploying plans.
>
> The world is fundamentally hostile. Life is a series of problems to be solved.

Let us call this the 'planning' view of everyday activity. Since I have defined it independently of the computer-program connotations of the AI notion of 'Plans', it is actually more general than the view handed down from Miller, Galanter, and Pribram. I wish to dispute this view of everyday activity and to substitute my own, the 'situated activity' view.

> Everyday life has an orderliness, coherence, and laws of change that are not the product of any representation of them. Everyday activity is almost entirely routine, even when something novel is happening.
>
> Everyday activity is fundamentally improvised; contingency is the central phenomenon. An agent conducts its everyday activity by continually redeciding what to do.
>
> The world is fundamentally benign. Life is a fabric of familiar activities.

People certainly use plans. But real plans are nothing like Plans. Rational, goal-directed activity need not be organized by a plan. And plans never serve as direct specifications of action. Instead, a plan is merely one resource among many that an agent might use in deciding what to do (*cf.* Suchman 1986, 1987). Before and beneath any plan-use is a continual process of moment-to-moment *improvisation.* Improvisation might involve ideas about the future and it might employ plans, but it is fundamentally a matter of deciding what to do *now.* Indeed, plan-use is a relatively peripheral phenomenon and not a principal focus of this work. (See Agre and Chapman 1988 for a different theory of plans. See also Chapman and Agre 1986.)

A1b What I did

Computational research into activity seeks technical ideas about action and representation that are well suited to the special requirements of situated, embodied agents living in the real world. The 'agents' could be robots we would like to build or creatures—insects, cats, or people—we would like to understand. To say that an agent is 'situated' is to emphasize that it is always in some particular situation in the physical and social world. It is always provided with particular concrete materials and involved with particular other agents. To say that an agent is 'embodied' is simply to say that it has a body. It is physically localized and has limited experience and finite abilities. It is always *in* the world and *among* the world's materials and other agents.

This report has three parts. This part, Part A, carefully defines the project and its method and vocabulary. Chapter A2 introduces a way of talking about organized activity based on the idea of *dynamics*—that is, regularities in the interactions between certain kinds of agents and certain kinds of worlds. I will argue that the principal task of artificial intelligence research is not the proliferation of complex forms of machinery but rather the elucidation of the dynamic structure of complex forms of activity. Having developed an understanding of dynamics, one should seek the simplest machinery that is consistent with the forms of activity one wishes to produce or explain. Chapter A3 uses a narration of the walk from my apartment to the subway to sketch some of the important dynamic concepts I have found useful in using analyses of human activity to motivate computational models. In particular, it introduces the notion of a *routine* and briefly describes some of the dynamics of routines and their evolution.

Parts B and C each describe an idea and a computer program that illustrates it. I will sketch these ideas and programs very briefly here. For more detailed summaries of the individual chapters in Parts B and C, see Chapters B1 and C1 respectively. In general, this report is organized to appeal to people with relatively little technical background. Those who wish to start with formal technical discussions should head for Chapter B5 and then proceed to Chapters C3 and C4.

Part B's idea is a way of organizing improvisatory action called *running arguments.* Whereas a Plan is executed by a dumb executive with little idea of the reasoning behind its actions, an agent engaging in a running argument can be far more flexible because

it continually rederives, to the greatest extent possible, the reasoning behind each moment's actions. This reasoning is non-monotonic and takes the form of an argument among conflicting options and considerations. Because the reasons behind consecutive moments' actions will usually be very similar, it suffices to continually incrementally update the arguments leading to action. To facilitate this incremental updating process, the agent maintains dependencies on its reasoning. Part B describes a computer program, the running arguments system, that illustrates these ideas. Driven by a rule set written in a fairly conventional rule language, the system interacts with a simple simulated world. The system accelerates its own operation by keeping dependencies on its reasoning. Chapter B5 demonstrates the running arguments system in action, carefully analyzing its performance and drawing some negative conclusions about conventional computational views of representation.

Part C's idea is a novel theory of representation for situated agents called *deictic representation.* Whereas a conventional objective representation scheme such as first-order logic individuates things in the world objectively (independently of the agent's location or heading or projects or attitudes) a deictic representation scheme individuates things indexically (in terms of their relationship to the agent) and functionally (in terms of the roles they play in the agent's ongoing activities). Thus deictic representation does not involve any kind of objective identity. While an agent might also use other forms of representation, such as natural language with its complex, socially organized notions of identity, I will argue that objective identity should not be a central concept in a representation scheme for situated agents. It is almost always a hindrance and hardly ever a help. This is a difficult idea. Part C illustrates it with a system called Pengi that plays a video game. The game requires its player to participate in organized, goal-directed improvisatory activity. Deictic representation permits Pengi to continually rederive its best course of action without complex machinery or elaborate computations. Part C concludes with some long, detailed stories about some of the ways in which a workspace helps organize everyday routine activities.

A1c Everyday life

In developing these ideas, my method was to move back and forth between observing real activities (involving both myself and others) and building computational models. My goal in watching people was not a shallow replication of human ways. I don't believe such a thing is even possible. Instead, I sought to understand the nature of activity in general. Exploring computational models—seeing what can and cannot be gotten to work—helps isolate which aspects of human activity are essential and which are not. If, as I believe, human activity is the way it is for a good reason, then that reason ought to be just as binding on robots as it is on people.

Everyday life is just the whole of our ordinary activity: making breakfast, gardening, hanging out, walking into town, cleaning up messes, shopping in supermarkets, working in offices, participating in everyday rituals. I propose investigating the everyday activ-

ities of human beings and relating the organization of human activities to hypotheses about the workings of human bodies and brains. While the logic of my proposal might be clear enough, it is difficult to properly appreciate the idea that everyday life is something that needs to be studied. Who is an authority on the nature of everyday life? It is a deep and consequential fact that simply living everyday life does not, in itself, qualify one to theorize about it. Second-hand intuitions and made-up stories need not, and usually *do* not, bear any relationship to the reality of everyday activity. One must go looking.

The hardest part of everyday life to see is its routine nature. Our simplest activities have a great deal of detailed organization: flexibly improvised adaptations and rearrangements, fluent synchronization, contingencies accommodated, and false starts rebegun, all taking place against a background of continual fine adjustments. Since the vast majority of this organization doesn't raise any obtrusive difficulties, it goes almost entirely unremarked. We don't we have many words for talking about it. We take it so thoroughly for granted that it is invisible even when it's staring us in the face. Even on a slowed-down videotape, it is remarkably difficult to see it without much patient effort.

When we talk about everyday life, what stands out to be talked about are *problems.* We complain about our problems; we focus on them and name them and discuss them and work out plans for dealing with them; we buy books and take classes and go to professionals to help us resolve them; and we create institutions whose job it is to alleviate them. Given this natural imbalance in attention, it is equally natural that, lacking detailed investigation, our ideas about the nature of activity should center on problems and their solution. Pretheoretical intuitions such as these suggest studying the mentation through which we plan our way out of problems. But such a view of everyday life has, to put it mildly, a warped sense of proportion. Its misplaced focus doesn't simply falsify the nature of everyday activity in the large, it also falsifies the nature of the particular phenomena it advises us to study. Problems aren't like that.

When I say that everyday life is almost wholly routine, I mean that it is *always* almost wholly routine. Even when we face problems, even when we're doing something new, even when we make mistakes, even when we're trying to pay attention and be careful, just about everything we do is something we can do routinely because we've been doing it so regularly for so long. In particular, to say that everyday life is almost wholly routine is *not* to say that our lives consist of stretches of mechanical execution punctuated by episodes of pure novelty. Everyday life *has* to be almost wholly routine or else it would be impossibly complicated. Even the activity called "solving problems" takes place against a background of routine activities: looking around, poking at your materials, trying things, drawing diagrams, asking help, keeping your balance, keeping your limbs under control, and so forth. Articulating the unproblematically beneficial structure of the routine background of our activities ought to change the way we look at *all* our activities.

Using more sophisticated ideas about activity to guide computational model-building is not simply a matter of knowing a correct, finished theory. Such things don't exist.

Instead, one must forever work at being aware of and articulating one's own experience of everyday life. This isn't a magical ability that some people are born with. And it certainly isn't a white-coated technology of introspection. It is a capacity that lies latent in everyone and awakens now and again by degrees with no particular correlation with age or class or abstract intelligence or moral exhortation or literature classes. Its basis is not theoretical abstraction but rather a spontaneous openness to the mind-blowing intricacy of humble phenomena. An awareness lost in projects and goals continually passes over these phenomena. To catch sight of them, one must, deep down, find one's own everyday life *interesting*.

So severe is the disjunction between prevalent computational methods and the actual nature of everyday life that it is hard to make any connection between them at all. I know people who, at home, can be perfectly articulate about the creative use they make of recipes but who, at work, insist that plans are like computer programs and treat any demurral as a heresy against reason. In trying to connect technical talk and everyday experience, it does not suffice to keep stretching quasi-technical words like 'planning' and 'knowledge' and 'hierarchy' to fit any next example that comes along. One must be willing to use technical difficulties—computational intractability, excessive complication, the seemingly incorrigible necessity for artificially constrained task environments, and generally the whole range of engineer's headaches—as opportunities to articulate the view of activity behind received technical ideas, to ask whether everyday activity is really like that, and to go looking.

Looking at everyday life can take many forms. It can involve watching people do things. It can involve watching videotapes a frame at a time. It can involve talking the phenomena over with people (preferably not technical people) who are especially articulate about their everyday lives. It can involve consulting the writings of people who have already looked carefully at everyday activity and then trying to see for oneself what they saw. While I have done all these things, for me looking at everyday life has largely taken the form of spontaneously noticing events in my own everyday life that seem relevant, often in obscure and even disturbing ways, to the technical questions I am studying. I will recite several stories about these events in the course of my presentation. You don't have to believe the stories if you don't want to. They are there to help connect abstract analysis to real experience. I hope they will provoke you to notice such things on your own. It is not that any list of stories can compel one to adopt one technical proposal rather than another. Instead, the stories serve a heuristic role, deepening the intuitions one brings to one's technical practice. The final test of engineering proposals is, as always, whether they work. Whereas the planning view of everyday life suggests machinery that doesn't work, the situated activity view, I believe, suggests different machinery that does.

A1d Technical and narrative language

Part of any conceptual reorientation such as the one I suggest is a redoubled effort to be clear about words and what they mean. The images called up by a vocabulary have enormous influence over our intuitions, and the coherent network of images called up by an unreflectively customary way of speaking can encode and convey an entire worldview, long before the first axiom is set to paper. This is equally true for the language of everyday life and for the languages of philosophy and psychology and computer science. In order to dig up the unarticulated assumptions behind our technical practice, we have to look at our words.

Many words will appear in scare quotes in this report. I should explain this practice. Words in double quotes are simple quotations. Words in single quotes, though, are words I would like to call into question. In doing so, my intention is never to ridicule those words or the people who use them. Instead, I want to call attention to the metaphors and stresses implicit in those words. These matters of image and connotation might be evident when words are used in poetry or story-telling, but they tend to hide when they're used in technical discourse. Recovering the history and associations of a quasi-technical word like 'structure' or 'control' can help us reconnect the intuitions behind the technology to the everyday experiences we also associate with those words. One can never make a knock-down argument out of such connections, of course, but they are regularly have heuristic value nonetheless.

I am particularly concerned to draw clear distinctions between the technical and vernacular—or, as I like to say, narrative—meanings of words. Plenty of AI's words have both technical and narrative meanings. It takes great care to avoid shuffling back and forth between the two meanings, blurring the vocabulary with which we tell stories into the vocabulary with which we program computers. One word that often suffers such treatment is 'plan'; Chapter A2 will discuss many more. In order to help avoid such confusions, I will capitalize the technical versions of a few especially important words—Plan, Problem, and Complexity—to mark the critical differences between their technical and narrative meanings.

Chapter A2

Machinery and dynamics

> "... the Western conception of the person as a bounded, unique, more or less integrated motivational and cognitive universe, a dynamic center of awareness, emotion, judgement, and action organized into a distinctive whole and set contrastively both against other such wholes and against a social and natural background is, however incorrigible it may seem to us, a rather peculiar idea within the context of the world's cultures."
>
> Clifford Geertz, On the nature of anthropological understanding, *American Scientist* **63**(1), 1975, page 48.

A2a Context and summary

Implicit in every community's way of speaking is some view of its world. This chapter has four purposes: to clearly distinguish between two broad views of human existence, *mentalism* and *interactionism*; to enumerate some of the ways of speaking that have made mentalism invisibly prevalent in AI and the greater Cognitive Science community; to explain why interactionism is the preferable view; and to describe how interactionism has informed my methods and strategy in performing the work reported here.

Section A2b defines and contrasts mentalism and interactionism. Each of these two -isms is organized by a certain taste in metaphors. Mentalism prefers metaphors of inside and outside. It begins with an entity called 'the mind'. Inside this 'mind' are mental structures and processes. These mental contents can be defined purely formally without any reference to the outside world. Inside/outside metaphors have governed AI research, almost without exception, since its beginning. Interactionism, by contrast, prefers such metaphors of interaction as servocontrol, participation, metabolism, and routine. A principled choice between mentalism and interactionism must begin with an awareness of the ways of speaking that tend to presume them.

Section A2c presents some of the methods of interactionism. Interactionist research begins with an investigation, both theoretical and empirical, of the *dynamics* of the interactions that arise between various kinds of creatures and various kinds of worlds.

An understanding of the dynamics of a given form of activity can help in inferring the machinery of a creature that engages in that activity—or in *designing* the machinery *for* such a creature. My own work has followed the principle of *machinery parsimony*: postulate the simplest machinery that is consistent with the dynamic phenomena you understand. The deeper your understanding of dynamics, the simpler the machinery becomes. My initial work has concentrated on some of the more fundamental dynamic phenomena, those involved in the everyday routine solitary activity of human beings. Future work must deal with learning, social interaction, and all the rest.

Section A2d explains why AI research needs dynamic theory. Any time you design a device, you have to explain why it ought to work. Such explanations often take the form of a demonstration that the device solves the general case of some formally defined Problem. I have nothing against formality. Solving the general case, though, almost invariably leads to a no-win trade-off between unrealistically simple formalizations and intractable computational Complexity. Heuristic methods are no help unless they come with some good reason to believe that the device will work. The only way out is through a sufficient understanding of the inherent orderliness of interactions between sensible agents and benign worlds.

Section A2e presents some distinctions that help in developing theories of the dynamics of human activity. The human machinery changes through learning, so it is best to distinguish between the innate, unvarying *architecture* and the *personality* that forms within it. The dynamics of your interactions with the world change, obviously, as your personality evolves.

Section A2f investigates the tangled question of the computational Complexity of conventional Planning as a study in the narrowness of mentalist theory. I would like to argue that conventional Planning makes a poor theory of activity for any existing or possible creature. Ammunition for such an argument would seem readily available in the form of extremely negative Complexity results. But Complexity theory itself, frustratingly, is an inappropriate means of evaluating theories of activity. Mentalist metaphors are responsible for the trouble in each case. Chapter B4 will analyze the question of Planning in a more sophisticated way.

Section A2g discusses about a dozen authors in other fields whose ideas about human activity have been organized around metaphors of interaction. They make a disparate group, but all of them offer useful suggestions about how one might proceed with the project of relating dynamic theories to computational principles.

A dynamic theory, like any other, is a large abstraction carrying no warranties against woolly thinking. This is the role of computer programs and observation of everyday life. Part A concerns only the form of the argument, not its substance. Programs and evidence will appear beginning in Part B.

A2b Mentalism and interactionism

In its technical practice, AI has a lot of vocabulary for talking about structures and processes inside of computers. If you start looking at the original vernacular meanings of these words, a remarkable pattern emerges. Actually two closely related patterns:

(1) A word that once referred to something in the world now refers to a structure in the computer. Common examples include 'situation', 'pattern', 'context', 'object', 'list', 'map', 'structure', and 'problem'. Individual programmers have defined hundreds of others.

(2) A word that once referred to an activity conducted by agents in the world now refers to a process occurring entirely in the computer. Examples include 'search', all verbs for operations on datastructures ('construct, manipulate, inspect, point at, traverse, collect, recycle'), and many predicates on technical entities. 'Efficiency', for example, was once a quality—i.e., not something to be measured—ascribed to a person.

This terminological practice has always been considered perfectly harmless. All those datastructures and processes are well-defined; they *exist* inside the computer; they need names; they bear a certain resemblance to those objects and activities; words' meanings are extended through metaphorical resemblance all the time; so why not? And, indeed, this is a perfectly valid argument. One has every right to define those abstractions and give them those names.

The problems start when these words become part of a theory of an agent's activity in the world. Many words have both vernacular and technical (or quasi-technical) meanings, and most of these meaning-pairs differ in just this way. I can't use the words 'search' or 'list' or 'stack' in an AI context without being taken to be talking about the insides of computers or heads. Things are even worse when someone writes a 'planner' one of whose arguments is a datastructure called a 'situation'; or an 'interpreter' one of whose arguments is a datastructure called a 'context' or an 'environment'. Students are exposed to this sort of talk without anyone systematically making the distinctions clear for them. This is bound to be confusing. What is the problem?

The villain of our story is *mentalism.* Mentalism, as I will use the term, is not precisely a scientific or philosophical position. Instead, mentalism refers to any psychology or philosophy organized around metaphors of Inside and Outside. According to mentalism, 'the mind' has 'contents' which are radically different from things in the 'outside world'. Mentalism is informed by a strong sense of boundary that marks off the mental as a separate realm. Above all, mentalism leads one to make theories that posit objects and processes residing entirely within the head. In particular, mentalism encourages us to define the topic of our research as a mental process called *cognition.* (For a clear statement of the form of mentalism most common in cognitive science, see the introduction to Fodor's *Representations* (1981). For an especially extreme statement that brings out much of mentalism's perverse inner logic see Fodor's "Methodological solipsism" paper, Chapter 9 of the *Representations* volume.)

We should pay attention to the place of metaphors in computational theorizing.

Metaphors suggest problems, judge solutions, define prototypes, and distinguish central from marginal phenomena. Above all, the metaphors behind our technical terms encode a view of the nature of activity. They also tend to perpetuate that view by making it seem transparently obvious. The metaphors of 'inside' and 'outside' are a shadowy backstage presence in AI. They can be seen in words like 'in' (the head or the mind), 'cognition' (a separate realm of inward activity), 'perception' (arriving inside), 'behavior' (emerging outside), 'correspondence' (between representations inside and the world outside) and many more words for mental events and things. The metaphors don't make you hold any particular technical and scientific position. The difficulties that arise can always be blamed on a specific position. The metaphors hover in the background, acting as arbiters of plausibility without ever taking any blame.

In disagreeing with mentalism, I am not denying a theory or a philosophy, though I disagree with plenty of mentalist theories and philosophies. I am not saying that people don't have minds (what an awful thought!) or denying the existence of cognition. Certainly we have machinery in our heads and certainly people often sit and think. The question is, as theorists, how do we proceed from there? I don't believe it is very useful to posit very abstract entities in the head, and certainly not things like search trees and sentences of logic. Most especially, I am not denying any phenomena. I am only criticizing a way of talking about them.

Mentalism provides a simple formula that provides plausible answers for all questions: put it in the head. If agents need to think about the world, put analogs of the world in the head. If agents need to act in situations, put datastructures called 'situations' in the head. If agents need to figure out what might happen, put simulations of the world in the head. If there are many possibilities, put a process of 'search' in the head. The tacit policy of mentalism, in short, is to reproduce the entire world inside the head: a 'world model'. Consider the slogan of 'mental models', according to which 'reasoning' about the world depends on having access to as complete a simulacrum as possible. Or the slogan of vision as 'inverse optics'. The development of computational methods over the last thirty years has been shaped by the steady pressure of this habit. The sophisticated structures and processes we've learned how to define are not geared to living in the world, they are geared to *replacing* it.

In place of mentalism, I prefer a view I call *interactionism*. Interactionist words—'interaction', 'conversation', 'involvement', 'participation', 'servocontrol', 'metabolism', 'regulation', 'cooperation', 'improvisation', 'turn-taking', 'symbiosis', 'routine', 'management', and so forth—shift our attention from 'cognition' to 'activity'. They lead us to posit structures and processes that cross the boundaries of agents' heads. Sure, perhaps some of these structures and processes *are* entirely inside of agents' heads, but that's just an unusual special case with no particular privilege. Future chapters will describe many interactionist theoretical ideas. Many more are needed.

Mentalism and interactionism are incompatible. No doubt many mentalist ideas have interactionist analogs and *vice versa*, but each of them offers its own distinctive way of approaching every phenomenon of human existence.

It is difficult to argue against mentalism. Not being a technical position, it doesn't admit of formal disproof. If interactionist technical proposals achieve stunning technical successes, that's certainly good evidence. Unfortunately, such successes in themselves are unlikely to form a basis for comparison between mentalism and interactionism. By making different aspects of human existence prototypical, mentalism and interactionism suggest starting in very different places. Until the far-off day when we finish a computational psychology, one can always ask 'how does your theory explain X?' This report is mostly about something that has rarely even been considered a problem: the organization of ordinary, routine activity. Its greatest weaknesses are in areas that have generally been considered central. In these areas I lack access to the easy answers of mentalism.

Fortunately, stronger arguments are available. These arguments concern the metaphors themselves. Let us return to the words AI uses for the insides of computers—'search', 'list', 'pattern', 'context', etc. Dozens of times I've given up on one of these ideas, only to discover that the idea I really wanted was very much closer to the original, vernacular meaning of the word. Gradually I realized that the vernacular words, unlike their AI namesakes, suggest interactionist ideas.

- An actual search is an interaction between an actual agent and an actual space (like the messy workbench where the tin-snips might be buried).
- You might accumulate an actual list (of, say, things to get at the store) on an actual piece of paper (like the back of the bank statement that has been cluttering your coat pocket).
- Many activities are organized in part by actual, physical patterns, for example sewing, filling out forms, and writing from an outline.
- It's hard to make a hard-and-fast rule about whether to cross the street since the wisest decision can depend on so many aspects of the context.

Later chapters will present many specific arguments about the metaphors behind mentalist ideas. In general we should ask, what troubles should befall mentalist research if the right way to talk about human activity is actually interactionist? Interactionism constantly directs our attention to the connections between inside and outside; it constantly reminds us that inside and outside are inextricably bound up with one another. If mentalist research always starts by drawing a sharp distinction and firm boundary between mind and world, we should expect inside and outside to try to reunite in some more covert way. But this is exactly what happens when mentalism tacitly pretends that the world is inside the head. The price of mentalism's artificial partition between inside and outside is the seductive but spurious plausibility that derives from constantly obfuscating the *difference* between inside and outside.

If I insist on restoring AI's current words to their vernacular meanings, what words is psychology supposed to use? Switching to neologisms would miss the point. There is nothing wrong with recruiting existing English words to name theoretical ideas. I'm

only disagreeing with the practice of indiscriminately populating heads or computers with analogs of the materials and equipment of the outside world.

In his famous discussion of the ant on the beach, Simon clearly anticipated the view that organized activity arises from the interaction between relatively simple creatures with complex but benign worlds (1970, pp. 24-25). But one page later, Simon abandons his insight, asserting that he is only interested in cognition and not in embodied agents, and moves on to discuss his studies of human performance on cryptarithmetic puzzles. The computational models of Newell and Simon's school have affirmed their wholly cognitive view of such puzzle-solving by locating the state of the process within the individual's short-term memory. (See Section C5d.) I do not know whether Simon meant to draw an equivalence between interacting with one's short-term memory and interacting with the outside world, but I regard these two concerns as polar opposites. To be fair, though, Simon has more recently participated in interesting research on "external memory" practices such as the use of scratch paper (Larkin and Simon 1987). (For other work on this subject see Intons-Peterson and Fournier 1986.)

All our thinking and acting and learning takes place against the enormous background of everyday routine activity. For this report, it will be enough to describe the network of patterns of interaction that make up this activity. Let us now begin developing a vocabulary for ideas about interaction.

A2c Machinery and dynamics

The central idea of this report is the distinction between machinery and dynamics. This section defines both terms, describes the notion of dynamics in detail, describes the close interconnections between machinery and dynamics, and explains the consequences of these ideas for interactionist research methodology.

A machine, as usual, is a physically realized, formally specified device. It is an object in the physical world that participates in the laws of physics. It will presumably have some internal state of its own, including perhaps the potential for changing its internal configuration. It's analog or digital or both. The mass term 'machinery' is intended to suggest particular metaphors for thinking about the machinery's physical realization: its operation depends on a configuration which is more or less fixed.

The notion of dynamics is less familiar. It concerns the interactions between an individual (robot, ant, cat, or person) and the world. The word is used in a number of forms. In making dynamic explanations one often isolates a particular 'dynamic', which is a common, lawlike regularity in the way that some sort of individual interacts with some sort of world. As an adjective, there are 'dynamic explanations', 'dynamic theories', and the like. One might speak of the 'dynamics' of Simon's ant on its beach and the 'dynamic structure' of the ant's life on the beach. Both simply mean "everything there is to say about the way the ant and beach interact." Both the version of dynamics that Newton and Euler would use to describe our physical interactions with the world and the version of dynamics associated that Freud would use to describe our emotional

interactions with the world are instances of my general notion, but neither of these extreme cases figures much in this report.

Here are some simple examples.

- In your kitchen cupboard you probably have a set of bowls stored in a stack. If you use one of them, you are likely to return it to the top of the stack. Over time, the bowls you never use (or no longer use, or have not begun using) tend to sink to the bottom. If you use some of the bowls and not others then the bowls you use and the bowls you don't use become segregated. These sinking and segregation effects are dynamics.
- The elevators in MIT building NE43 are not very smart. If someone is waiting, say, to go down from the fifth floor, then all elevators that happen to be passing that way will stop. It is common for three of the four elevators to stop, and I have seen all four. As a result, if you are in an elevator and it stops somewhere on its way to your destination, it is quite likely that nobody will get on. Having learned this, and having often waited too long for elevators that were off stopping uselessly at other floors, many residents of the building have formed the habit of punching the DOOR CLOSE button after only the most cursory search for approaching people. As a result, many visitors have noticed and commented upon the seeming rudeness of the building's residents. This has gone on for the whole of the nearly ten years I have worked in this building, despite its continual turnover of personnel. This interaction between the elevators and the building residents is a dynamic.
- I can't work without music playing, so I have a record player and a shelf of records in my office. Since I play upwards of two dozen records a day when I'm working, I've developed a definite routine for getting a record down off the shelf, removing it from its sleeve, picking a side to play, putting it down on the turntable, cleaning it and setting it going, removing it from the turntable when it's done, returning it to its sleeve, and returning it to its place in alphabetical order on the shelf. Though this routine happens very quickly (about fourteen seconds to put the record on and about twelve seconds to put it back), it is enormously complex, involving several changes of grip and orientation. One evening I went through this routine about thirty times and wrote down every detail of it. In doing so, I was able to satisfy myself of the accuracy of the following hunch about it: if I always play side 1 (say) of a given record, then side 1 will always be facing up at the point in my routine where, having just removed the record from its sleeve, I check whether the side I wish to play is facing upward or downward. This is unfortunate; since I happen to be holding the record from underneath just then, it would be much less clumsy to turn it over in the course of putting it down on the turntable. This invariant in my interactions with my office is a dynamic.
- This story covers four days one winter. One morning when arriving in my office, I decided I was sick of my coat cluttering my office, so I decided to leave the coat lying on top of the file cabinet just outside my office door. Shifting my

concern to the day's work, I walked into my office and tossed the door shut behind me as always—except that today it didn't slam behind me as always. Huh? Investigating, I found an edge of the coat caught in the door jamb and preventing the door from closing. I herded the coat out of the way, closed the door, and went back to work. The next day I left the coat on top of the file cabinet, headed into my office, and tossed the door shut as always—and it didn't slam behind me again. Shoot. This time, though, I immediately knew what the problem was. The next day I left the coat on top of the file cabinet as before, but as soon as I turned to head into my office I realized that the coat was liable to get caught, so I herded the coat out of the way again. The fourth day, I was aware of the problem as I was placing the coat down on the file cabinet, so I made a point of placing it as far as was readily practicable from the door jamb. On each day, a bit of the previous day's insight had drifted back toward the beginning of the routine. This backward transfer effect is a dynamic, one of the principal dynamics through which routine patterns of activity evolve.

These examples illustrate a number of points.

1. Dynamics are only descriptions, not causal agencies. The bowls do not sink and segregate because the dynamics force them to; they do so because on a series of occasions you returned the objects you've used to the top of the stack. Everyday activity is, as the ethnomethodologists say, 'locally organized' (Heritage 1984). Nonetheless, just as a figure of speech, it is often useful to say that something happens 'because' of some dynamic.

2. A dynamic is most emphatically not a structure in any agent's head. A dynamic is a theorist's description, not a datastructure or a plan or a mental object of any kind. Having identified a recurring form of interaction between an agent and its world, one can set about determining what kinds of machinery might be compatible with it. The agent's machinery *participates in* the dynamic but is not solely *responsible for* it.

3. The dynamic depends on certain facts about both the individual and the world. It depends on the world in that stacks require gravity, objects are often made to stack, stacks tend to stay orderly unless disturbed, organic garbage decomposes, customs propagate, my records are at eye level where my turntable is at waist level, doors close flush to their jambs, and so on. These particular dynamics only depend on some rather vaguely described aspects of the individual (puts the stacked objects back reliably, participates in customs, checks record labels). Many others depend on more specific aspects of the individual's machinery, either how it works or how it's configured. It is an interesting question, one which I cannot now answer, whether the backward transfer dynamic would occur with any agent whose learning machinery works in any sensible way. A dynamic description is not simply a description of an agent's outward behavior. A dynamic describes a recurring

causal chain in which components of the individual's machinery participate on an equal basis with objects in the world.

4. A dynamic only continues occurring *ceteris paribus*; many different events might interrupt it. You might use all the bowls in a stack for a party one evening so that the next morning the newly reconstituted stack will be scrambled. The kitchen might be closed. Someone might play my records in my absence. My file cabinet might get moved. In studying an activity one concentrates on its more stable dynamics. But any dynamic description that would aspire to the status of natural law is limited by the possibility that any of a thousand additional factors might arise to change the outcome next time.

5. Most likely these dynamic effects are unintended; usually they are unnoticed as well. If they were intended or noticed, of course, they would still be dynamics. It is common to notice and describe to yourself an existing dynamic effect. The dynamic picture might then become more complicated if you begin deliberately doing things differently or if you go out of your way to encourage or retard the effect. Having noticed your stack of bowls segregating, you might deliberately put a rarely-used bowl back in its place to make the commonly-used ones easier to remove. Building residents might notice the prevalent pattern of rudeness and resolve to be more polite. Noticing the backward transfer dynamic in action definitely accelerates it, though I don't understand why. What's more, once you articulate a dynamic effect, the effect will often spread to new situations, even without your specifically intending it to.

6. Activity in workspaces (kitchens, shops, desks, bathrooms, cars) tends to have a great deal of dynamic structure. These places are also where our most complex solitary activities take place. They are thus important laboratories of dynamic theory. (For some wonderful stories about the dynamics of the use of appliances and tools in workspaces and how the implications of these dynamics for design see Norman 1988.) One central question is, how is it that we take so much advantage of the dynamic structure of workspaces without representing most of it?

7. A dynamic description does not completely specify any particular workspace or any particular episode of bowl-using or elevator-riding or record-playing or coat-stashing. No matter how carefully you write out a description of an instance of human action, I and many others have found through long experience, you will always find yourself with more to write. Indeed, the very action of writing out a round of description is generally enough to make you see a whole new dimension to the action that you had previously overlooked. I don't know why this happens, but it does. Fortunately, there is no need for completely specifications. We only need enough dynamic theory to sufficiently constrain a theory of machinery.

How can ideas about dynamics help us find ideas about machinery? Accounts of the dynamics of everyday life play two roles, one general and one specific. The general

role is to impress us with the inherent orderliness of everyday life and thus make us more confident in relatively simple accounts of machinery. The more specific role of dynamic descriptions is to help constrain particular aspects of the machinery. Any machinery issue (such as timing, state, speed, digital *vs.* analog, or noise tolerance) makes predictions about dynamics. Once you have a dynamic question in mind, you can consult the reality of everyday life. Indeed, observations of everyday life should be an everyday part of computational research into any aspect of activity. If you are preoccupied with some technical question, often you will simply spontaneously notice what you need to know as you're making breakfast or taking out the trash. Or you might watch somebody else's activity or look at a videotape.

In working backwards from observations about dynamics to hypotheses about machinery, it helps to imagine that someone explicitly designed the machinery for everyday routine activity. What was the designer's reasoning? What dynamics do various sorts of machinery get themselves into, how do these compare to the dynamics we observe, and what design principles do these comparisons suggest? The answers to these questions will raise new questions which occasion further observation. Repeat this cycle of theory and observation a hundred times and you'll be ready for rigorous experimental tests. I've been through it on two large occasions and several small ones. The first large occasion produced the idea of running arguments (see Part B), the second the idea of deictic representation (see Part C).

An account of the dynamics of everyday life explains what the machinery is responsible for. The machinery, in turn, determines what accounts of dynamics can be explained. Machinery and dynamics constrain one another so strongly that accounts of machinery and dynamics should be developed in parallel. Otherwise, one will inevitably be misled into positing useless machinery and unimplementable dynamics. A contrary view insists that "you can't design the machinery unless you know what it's going to compute." Such a slogan tends to emphasize the virtues of generality and explicitness and discount the constraints of physical realization. This is a mistake. One might formulate the what's-computed in many ways, and today's neglect of implementation issues brings tomorrow's baroqueness and intractability.

The most important principle of interactionist methodology is *machinery parsimony.* Postulate the simplest machinery that is consistent with the dynamic phenomena you understand. When you feel the need for extra machinery coming on, go look at the phenomena and ask what people really do. Do this even if your goal is engineering and not psychology: if people don't do something, they probably have a good reason not to. Rigorously applying this principle, we will discover repeatedly that *the deeper your understanding of dynamics, the simpler the machinery becomes.* This is much preferable to the more common approach of automatically ascribing any regularity in human activity to explicit representations and general algorithms residing in the head, leaving 'efficiency' for someday.

Why does an understanding of an agent's interactions with its world lead to simpler hypotheses about its machinery? Real agents *lean on the world.* The world is its own

best representation and its own best simulation. It isn't an obstacle or a problem, it's a helpful place. Your interactions with the world, both past and present, provide many ways to alleviate computational burdens. Why conduct elaborate deductions about your surroundings when you can look and see? In particular, why maintain elaborate control structures when you can look and see what needs to be done? Why make highly detailed Plans when you can improvise? Why require instant expertise when you can improve by just keeping on doing it? Why try figuring it out yourself when you can collaborate with others who have been there? Why insist on figuring out every situation afresh when you can trust your accumulated experience? All these dynamic phenomena can work together to suggest imaginative ways to simplify machinery or even eliminate parts of it altogether. Put someone in solitary confinement and they fall apart: people rely on the *organizing presence of the world.*

I am suggesting an inversion of values, not only in artificial intelligence but in all forms of computational and psychological inquiry. Faced with an empirical phenomenon to explain, our first explanatory recourse should be to dynamics, not to machinery. Faced with a technical problem to solve, our engineering should begin with dynamics, not with machinery. Heretofore, people have gotten prizes for inventing new machinery. But we've got far too much machinery. I would like to suggest that people get prizes for *getting rid of* machinery. One should aspire to invent novel dynamic effects and experience regret when forced to invent novel devices. In Parts B and C of this thesis I am going to state, dead seriously, that plain, ordinary combinational logic suffices to support some important dynamic phenomena. But combinational logic is obviously not the critical contribution of my work. If I were inventing simple machinery for its own sake, I would be wandering in mechanism space. And if I were mindlessly applying my ideas about machinery to every next problem, I would be missing the point. My contributions are my description, admittedly sketchy and provisional, of some of the dynamics of everyday routine activity and my suggestions about how a particular type of simple machinery is capable of participating in these dynamics.

Where should we start? Psychology must account for a wide range of phenomena, but research must find some principled way to focus its attention. Focusing on a particular domain, in itself, usually doesn't isolate a research problem, since most dynamic phenomena operate in most domains. In other words, any given activity is just as complicated as activity as a whole. One might also try focusing on one module of machinery at a time. This works fine for peripheral faculties like vision but there are good reasons to believe that the central system isn't and couldn't be modular (see Fodor 1983).

My own approach is to focus on particular dynamic phenomena. In this report I will consider some of the dynamics of everyday routine activity. Everyday routine activity is a promising place to start because it is the most pervasive and representative sort of human activity. Every activity is built on a base of unproblematic routine. By studying routine activity, I start studying everything. Everyday routine activity is also the sort of activity our cognitive machinery was designed for. Studying everyday activity encourages a realistic picture of everyday life as benign and orderly. It no longer seems

necessary to invoke general-purpose methods at every turn.

I will further restrict my attention in three ways. First, I will only consider solitary activities. (See Chapman (forthcoming) for a start on cooperative activities.) Second, I will consider only adulthood and not development. Third, I will only attempt to explain the steady state of routine activity. In other words, I will concentrate on the dynamics that don't involve learning. Even with these restrictions, we can make some good guesses about architectures. I will offer several initial suggestions about the dynamics of learning, but I will present no actual learning scheme. Lacking detailed theories of the dynamics of personality development, I will play programmer throughout, building whatever structures seem to work within principled limits.

A2d Why be concerned with dynamics?

There are several reasons to be concerned with dynamics.

The dynamics of everyday life describe what our cognitive machinery *does*: its operation is a critical part of our purposeful activity in the world. Simple as this claim sounds, though, its actual substance compared to existing AI practice is subtle. Method-conscious AI people already claim to work from a specification of what their devices are to do. At the level of the theorist's design of an agent's machinery, the machinery is typically broken into modules, each of which solves (even if heuristically) the general case of some formal Problem. At the level of the agent's concrete actions in the world, these actions are typically specified in terms of a Plan, that is, an explicit representation of primitive actions to be performed and their expected consequences.

(I've capitalized 'Problem' because, as with 'Plan', it's important to avoid confusing them with the ordinary word. The capital-P notion of 'Problem' is the theory-of-computation notion of a function from discrete inputs to discrete outputs, not the notion that comes with the slogan of 'problem solving'. I also disagree with problem solving as a view of action, but that's a different matter. See Section C5d.)

Both Problems and Planning suggest misleading metaphors of activity. Each of them concentrates on the boundaries of a module. A problem or goal arrives, thinking takes place, and a solution or Plan emerges, eventuating in a stretch of action. Improvisation, contingency, feedback, or midstream changes have no place in this picture. Nothing happens that the agent doesn't make happen, and nothing consequential is true that the agent doesn't represent. Focusing on the action within a self-sufficient module makes it easy to forget the outside world altogether. To put the world back in the picture, we have to understand what really goes on in the world. The aspects of the world that matter are the dynamic ones: that things tend to stay put, that habits have cumulative consequences, that you are more likely to notice things that are out in the open, that one automatically remembers how to ride a bike but doesn't automatically remember people's names, that if you keep on consuming a resource it runs out, and so on. Dynamic matters have their own laws; these laws would be hard to express by talking only about machinery.

Disregard of dynamic matters leads to two forms of confusion that may seem to cancel but in fact are tenaciously cooperative. Without any sense of what an agent in the world does and doesn't have to do, one must assume the worst and design algorithms that can solve the general case. But that same ignorance also allows oversimplifying metaphors to seduce one into trivialized formalizations of the issue at hand. The more intractable one's formalizations become, the less motivation there is to adopt more realistic ones. One example of this destructive symbiosis is general-purpose Planning. A theorist designing a general-purpose Planner has no way of knowing what cases are actually important to get right because the criteria are infinitely variable and can only be known when the time comes. As it turns out, general-purpose Planning is intractable or even undecidable for any but the most watered-down formalizations. (Section A2f will discuss these results.) Planning as it is generally formalized cannot model the full complexity of real human activity, but only the simplest possible formalizations seem technically practical. (Section C5c will discuss some particular proposals in more detail.) In short, general-purpose Planning is a conceptual black hole attracting anyone who neglects to work out the dynamics of activity the world of their interest.

Similar comments apply to attempts to study individual 'domains'. Restricting oneself to a single domain, like chess or medical diagnosis or stacking blocks, is no help because all nontrivial human activities appear to exhibit most of the important dynamic phenomena. As a result, it is a rare AI project that doesn't make a mockery of its domain, isolating a real computational issue only by boiling the domain down to a formal puzzle. To salvage any methodological respectability one must solve this puzzle in the general case, the same black hole.

The origin of these black holes is the very attempt to view situations in everyday activity as 'cases' of a 'Problem'. One must always ask, which dynamics are actually needed, and which actually occur? When you are faced with a real live problem (in the ordinary sense of the word), you have many advantages over a Planner or other general algorithm. You have a past which has provided you with relevant experience. You have a future which goes on whether you succeed or fail, provided you don't get yourself killed. You have all the resources of the situation to help inspire you to solutions. Most importantly, you know why you care about this problem. You can judge what's good enough, you can decide to take a different tack, or you can decide to give up and move on. A general-purpose algorithm can't make judgements like these because there is no *a priori* limit on what might be relevant to them. The usual result is a simplifying assumption that no such judgements are to be made, or only a very narrow, restricted class of them. The resulting general problem is almost invariably intractable; thus the black hole.

Getting out of these black holes requires an account of the dynamics of everyday routine activity as a whole. To understand the problem, it helps to think of all technical difficulties in AI as falling into two classes, OR problems and NOT problems. OR problems come from the need to anticipate all contingencies through search. When either X or Y might happen, one must search through both and risk a combinatorial explosion.

NOT problems come from the all-too-familiar gap between knowing something to be true and merely having failed to prove it false. Attempts to build general-purpose Planners encounter both OR problems and NOT problems, as will any general-purpose algorithm. Somehow an agent acting in a concrete situation must have a sufficient grasp of the totality of what-might-be-going-on that general search is unnecessary and a policy of acting on any idea that isn't obviously wrong won't be fatal.

A2e Architecture and personality

The machinery is made of an innate architecture and an acquired personality. After defining these terms I will discuss how they relate to issues of dynamics.

The word *architecture* used to concern buildings; here it comes from computer design. Architecture is the machinery you are born with. A theory of architecture should explain what sorts of things are in the machinery, what they do, how they interact, what happens in serial and what in parallel, and roughly how fast things happen. Many standard computational design issues arise: whether there are latches, whether there is a clock, whether there is persistence and delay in the circuitry, how new connections get made, and whether everything can really be connected to everything else.

Most existing computer architectures are specified in terms of their instruction sets; the machine interprets instructions and has a many-layered memory hierarchy. The kind of architecture envisioned here is quite different. It has two regions, the (modular) periphery, which concerns low-level perceptual and motor operations, and the (nonmodular) center, which concerns everything else. I'll assume the picture of the periphery outlined by Marr (1982) and Fodor (1983). In particular, the boundary between periphery and center is clearly defined. Our main topic is the center, for which modular methods are entirely inappropriate.

The possibility of distinguishing the center from the periphery does *not* imply that we can study it in isolation. The dynamics of an agent's involvements in the world implicate the periphery and center equally. Since concrete activities place the periphery and center in constant interaction, the design of each of them strongly influences the other. Part C will pursue this theme.

Personality, as I'll use it, is not the vernacular word but rather a computationalized version of the psychoanalytic term. Your personality is the structure that gets built within the central system of your architecture in the course of your life's activities. The presumptions of this definition (that stuff gets 'built' in your head, that it is usefully spoken of as 'machinery' and as having a 'structure') are part of the burden of my argument. Your personality, according to this argument, is a large network that is accumulated by a dependency-recording mechanism (see Part B). The word network is ambiguous: this is a network in sense of a physical device, not in the sense of a datastructure (as in the phrase 'semantic network'). The theorist inquires into the structure of the network. Of course, the network can vary among individuals and be a mess in all of them. But in the light of dynamic theory there is much to say about

personality structure, on both large and small scales.

Architecture and personality both interact strongly with dynamic issues. By working out enough of these interactions we can hope to arrive at a finished theory of architecture. The simpler the architecture the more hope there is for an early victory. So far I've been able to do a great deal with only simple assumptions about architecture, but obviously many issues remain.

Details of architecture design have dynamic consequences. Much of the present argument about architecture proceeds directly from claims about dynamics, though it soon becomes necessary to take aspects of personality structure into account. Given any new phenomenon to explain, it is always tempting to postulate a new special-purpose architectural feature. But machinery parsimony requires an honest hunt for possible dynamic explanations that involve little or no modification to the aspects of architecture you already believe in. For example, often one can explain the phenomenon by making some assumption about personality structure. In this case you must search for dynamics that might give rise to such structures.

Once the outlines of architecture have been settled, matters of personality have their own intricate dynamic consequences. How you live your life depends on your personality and on the world, and your personality depends on how you have lived your life in the past. This dialectic is exceptionally important for infants and small children because they are laying down the most basic aspects of their personalities. The dialectic is probably simpler in adulthood. We might idealize it as follows. Most of the time your life is in equilibrium. Nothing new is happening and there is an established, routine way of dealing with every situation. Then something new happens: a novel opportunity, discovery, contingency, or responsibility. In dealing with it you do something new, augmenting your personality with the new skill and thus modifying the dynamics of your life. Your new action will generally bring about another novel opportunity, discovery, contingency, or responsibility. This cycle continues until your life achieves a new equilibrium.

For example, someone gives you a package of loose tea. To use it, you have to acquire the right equipment. Then you have to learn how to use it through a series of firsts: first time opening the package, timing the brew, washing the equipment, putting it away, finding it again, discovering its quirks. Your routines for using it evolve. Eventually you get it down. The value of dynamic descriptions is evident here: as a result of a single event, your machinery is modified in dozens of separate ways whose interconnection is only comprehensible in terms of the dynamics of tea-making.

A2f Complexity and efficiency

It would be nice to have some principled method for evaluating computational theories of action. The most widely used technical vocabulary for comparing the efficiency of different computational methods is the computation-theoretic notion of asymptotic Complexity (Tarjan 1987). Complexity theory can seem perverse to the uninitiated. A

brief review will help remind us why the uninitiated are right. In particular, we will see how poorly suited Complexity theory is for discussing the appropriateness of an agent's machinery for its ongoing life in its world. Computational Complexity is defined with reference to a mathematical entity called a Problem, which is a class of formal input-output pairs. It envisions a machine engaged in a stylized dialog with an interlocutor. The interlocutor presents the machine with an input and waits until the machine replies with a corresponding output. Typically the interlocutor requires that the output be the exact output specified for that input by the Problem. In this case we say the machine 'solves' the Problem.

The sequence of queries is arbitrary. The machine cannot maintain any internal state between queries. Typically, the machine will provide the same output every time a given input appears. If not, such as when the machine employs some randomizing device, there is no significance to which particular output the machine returns on a given occasion.

(Incidentally, I capitalize Complexity because, notoriously, it bears little relationship to any vernacular notion of complexity. Indeed, the simplest method is often the most Complex and the least Complex method is often very complex.)

Complexity theory is concerned with some properties of this dialog. The best-known theorems concern 'asymptotic worst-case time Complexity'. This is a measure of how long the machine takes to return an output as a function of the 'size' of the input. (What aspect of the inputs should be accounted as their 'size' is often a topic of dispute.) The theorems concern the behavior of this function as 'size' grows infinitely. To make sense of this concern, one imagines the machine to have an infinite supply of 'memory' or 'tape' or, in the case of parallel machine models, 'processors'. Given a Problem, one defines the Complexity of a Problem as the best Complexity function one can obtain across all machines in some class that solve that Problem. It is typically accounted a positive result if this function is bounded by some polynomial in the input size, and a negative result if this function is exponential in the input size. For example, a Problem would be exponential if, by adding one new element to the input, one could always double the amount of time required to produce the required output. A polynomial Problem is said to be tractable and an exponential Problem is said to be intractable.

The view of activity envisioned by the metaphors of Complexity theory is mentalistic. A definite boundary separates the machine from its interlocutor. Only the occasional input or output crosses this boundary. One discusses processes that occur entirely within the machine. Beyond the constraints on single isolated input-output exchanges, one cannot discuss any significant property of the dialog between machine and interlocutor. The machine has no internal state and thus no continuing identity. The machine cannot refer back to its interlocutor, much less to some 'world outside', in the course of its deliberations. If it cannot produce the required output given only the required input then it simply fails. Thus the input must encapsulate all the information that might be relevant to the machine's reasoning. Indeed, in AI a Problem's input often includes something called a 'situation', as if a situation were something one could 'pass in' to a

machine. Considerations of the Complexity of an AI algorithm become a function of the 'size' of a situation, whatever that means. These properties of Complexity theory bear little relation to the job of being an agent in a world.

In 1985, Chapman proved a series of Complexity-theoretic results about conventional nonlinear Planning. A Planner takes a (description of a) situation and a (description of a) goal as inputs and produces a Plan as output. The Planning Problem specifies a mapping between inputs and acceptable outputs, that is, correct Plans. Input size is reckoned in terms of the sizes of the situation and goal description. The Planner is organized as a nondeterministic search through the space of partially specified Plans. As soon as this search finds an acceptable fully specified Plan it returns. The algorithm's inner loop is the subroutine that checks the partially specified Plans. Chapman asked about both the Complexity of the checking subroutine and the Complexity of the nondeterministic search as a whole. These Complexities depend on exactly how one formalizes situations, goals, and Plans. There appears to be no way to guarantee that the search itself will succeed, even when a correct Plan exists. This has not been a problem in the simple cases that have been tried. More striking is the number of assumptions required to make the Plan-checking inner loop itself tractable.

- Atomic actions.
- Atomic domain propositions.
- No loops or branches in the plan.
- No derived side-effects.
- Effects may not depend on the situation of action.
- No resource constraints.
- Goals are conjunctions of atomic propositions to be achieved.
- No autonomous processes.

Each of these assumptions, once interpreted in terms of ordinary activities, is violated all the time in everyday life. But eliminating any one of them makes Plan-checking intractable. (Most of these results are described in Chapman 1987, which explains the jargon. The rest are personal communications from Chapman and easy to rederive given the already published proofs.) Few interesting tractable relaxations of them are known.

These are, by conventional standards, extremely negative results, about as negative as one could imagine proving. At first Chapman and I were both very pleased with these theorems. We knew there was something wrong with Planning and we hoped to parlay his results into a general argument against it. It soon became clear, though, that they are of little value because asymptotic worst-case time Complexity has almost nothing to do with AI.

- Nobody is expected to plan themselves out of a worst case. One would like to speak of 'average case' Complexity, if only one could find a defensible way to define the 'average case' of Planning. We would want to weight the different Planning situations by the frequency of their occurrence, or their importance, or something like that.

- Heuristic methods and the attendant occasional screw-ups might be permissible if they bring tractability, if only one could find a defensible way to define 'occasional'. It seems likely that assessing the seriousness of a screw-up might require a great deal of knowledge about the situation and about what other considerations might be active.

- Approximately correct Plans might also be permissible if they bring tractability. Again, one needs a defensible way to define this 'approximate correctness'. It seems likely that assessing degrees of correctness also require a great deal of knowledge about the situation and about what other considerations might be active.

Upper-case Planning and upper-case Complexity have a conspiracy going. Upper-case Planning is an upper-case Problem. One rates Problems according to their Complexity. But negative Complexity results don't seem capable of definitively refuting Planning as a way of understanding the organized nature of human activity. This isn't fair. What's going on?

The original problem, common to Problems, Planning, and Complexity alike, is the inappropriate metaphor-system of inside and outside, boundary and contents. Activity in the world is just not like that. In the context of Complexity-theoretic analyses of Planning, the artificiality of mentalistic metaphors manifests itself in the need to 'pass in' the entire situation to any process of making or evaluating Plans. As so often, an artificial effort to reconstruct the world within the agent's head must compensate for mentalism's artificially rigid boundary between inside and outside. In practice, this tactic entails domain representations that cut a situation's description to a minimum. Ordinary activity is not amenable to such rigid *a priori* circumscriptions of relevance.

A2g Interactionism abroad

Systematic application of themes of interaction, participation, and activity is a novelty in the technologically oriented human sciences, but not elsewhere. I have learned a great deal by reading interactionist literatures in other fields. All of this work is grossly incommensurable with existing computational vocabulary. Nor do these projects represent any kind of unified movement; indeed they regularly disagree with or ignore one another. Moreover, most of their central figures would be wary about the use to which I am putting them here. Nonetheless, their shared themes offer a powerful alternative to the mentalist metaphors that burden artificial intelligence and cognitive science. Authors working within or inspired by cognitive science have made other, conflicting interpretations of some of them, but as you might expect, I regard most of these cognitivist appropriations as superficial or mistaken.

These writers are strikingly original thinkers and do not fall into helpful categories, so let us consider them alphabetically. I will try to summarize their work briefly though this is an impossible task, comparable to briefly summarizing literary criticism or chemistry

or Buddhism. All of them were active in this century or are still active today. The list is incomplete and omits many important figures.

Eric Berne was an American psychoanalyst who founded a school called *transactional analysis.* Although he saw his work as a development of Freud's formulations of psychoanalysis, he stated most of his important ideas in plain English in his best-selling books *Games People Play* (1964) and *What Do You Do After You Say Hello?* (1972). Insofar as Berne's books effectively founded the genre of popular self-help psychology, he is often not taken seriously. This is unjust. Berne mapped the recurring patterns of interpersonal entanglement he called *games*, describing what made each participant willing to participate in them. Berne developed his ideas into a markedly democratic form of group therapy based on using observations of individuals' unconsciously habitual games as ways of identifying and facing their underlying conflicts. Though some of Berne's particular formulations have dated with the growth of feminism and of object-relations psychoanalysis, his methods have had a wide influence.

Mark Bickhard is a psychological theorist who has developed a distinctive approach to psychology he calls *interactivism.* Although my attention was drawn to Bickhard's work only in the last year, some of his interests bear a striking resemblance to my own. For Bickhard, representation is interactive and functional and knowledge cannot be thought of as having the sort of structure we associate with symbolic programming (Bickhard and Richie 1983, Campbell and Bickhard 1986). Bickhard does not connect his work to computational themes, but this would be a interesting project.

Wilfred Bion was a British psychoanalyst who moved back and forth between the British object-relations school of psychodynamic theory and his own distinctive theories of group dynamics (1970). Expanding on the psychoanalytic principle that one's personality is organized through one's formative interactions with other people, Bion stressed the analogies between unconscious experience and the processes that arise in groups, be they families, circles of friends or colleagues, or therapy groups. He stressed the success or failure of a group in 'containing' the potentially disintegrative stresses experienced by the individual and he connected this theme to the individual's formative experiences of containment. The only accessible introduction to Bion's ideas I've encountered is (Hinshelwood 1987).

Harold Garfinkel is an American sociologist who founded a school called *ethnomethodology* (1967). While passionately concerned with the classical sociological problem of the nature of social order, ethnomethodology starts from a radical critique of the nature of sociological theorizing—and, by extension, of all theorizing about human activity. People predicate their actions on the existence of companies or parking spaces or rules or plans. We can study those actions but to endow those companies or parking spaces or rules or plans with any objective existence would miss the whole point. All those things are collective fictions kept alive from moment to moment by the cooperative actions of individuals. For Garfinkel, this phenomenon recommends a severe methodological particularism that is both constraining and liberating in ways that are difficult to express. Garfinkel's writing is difficult, and for good reasons, but (Heritage 1984) is an accurate,

clear introduction. Suchman (1987) has been productively practicing ethnomethodology in the analysis of computer-based office automation systems that use AI concepts.

James J. Gibson was an American perceptual psychologist who reacted against the prevailing notion of visual perception as the extraction of information from single, isolated retinal images (1985). Instead, he proposed a theory of *direct perception* according to which the visual system evolved to pick up *invariants* of the physical world over time (1979). He also described objects in the world as bearing certain *affordances* which enter into their characteristic patterns of sensorimotor interactions with human beings (for example, handles afford grabbing and pulling). Gibson's theories have been heavily criticized within computational vision research for their lack of a clear computational foundation (Ullman 1980), but computational theorists have paid less attention to his useful insistence that vision research start with environmental invariants and recurring forms of interactions between agents and their surroundings. For a brief account of Gibson's ideas see (Hagen 1985).

Martin Heidegger was a German philosopher who based his profound revision of Western philosophy on a careful description, in his 1927 book *Being and Time*, of the experience of engaging in everyday routine activity. He attempted, with substantial but incomplete success, to systematically reject the opposition between a perceiving subject and an independent external object and the attendant problems of epistemology and ontology. Instead, he described our experience of things as fundamentally bound up with their role in our ongoing projects. He also emphasized that our experience in the world merges indistinguishably with that of our neighbors and that our practices for getting along in the world merge indistinguishably with the traditions handed down through our culture. Heidegger's philosophy was the principal basis of Dreyfus' analysis of AI in his unfortunately titled book *What Computers Can't Do* (1979). Dreyfus views AI, correctly I believe, as having developed within a tradition largely unaffected by Heidegger's thought and the rest of Europe's twentieth-century innovations in philosophy. For a careful and sensible analysis of Dreyfus' arguments see (Preston 1988). Heidegger's writing (at least in Division I of *Being and Time*, which is the relevant text for our concerns here) has a crystalline precision that is hard to comprehend unless you already have some idea what he's trying to do. (Dreyfus forthcoming) is an excellent guide to the text.

Kenneth Kaye is an American developmental psychologist whose book *The Mental and Social Life of Babies* carefully describes the evolving dynamics of interactions between infants and their parents, starting from the very simple interactions involved in feeding and into the critical ability, seemingly specific to human beings, to take turns in play and conversation. Once these turn-taking dynamics are in place, they form a stable foundation on which much more complex patterns of interaction can be built. Kaye emphasizes that the evolution of these dynamics depends critically on the parent regarding the child as more intelligent, rational, knowledgeable, goal-directed, comprehending, cooperative, *etc.* than the child really is. Unfortunately, his theorizing is limited by the primitive theories of action and communication he has available to

describe the phenomena he has observed. (His theory of action, for example, is taken from Miller, Galanter, and Pribram.) I expect that more sophisticated ways of talking about activity can help turn Kaye's observations into a useful proposal about how an infant's learning machinery can exploit the interactional regularities Kaye observes.

Jean Lave is an American anthropologist whose studies of everyday cognition have been heavily influenced by Soviet activity theory and exhibit a theoretical depth and rigor that is rare in this country. Her book *Cognition in Practice* (1988) explores the peculiar fact that 'just plain folks' who score poorly on school arithmetic tests and consider themselves bad at math perform very well on the complex calculations required to shop in the supermarket. She rejects the view of cognition as something that takes place in the head. Instead, she view cognition as a concrete activity that takes place in the individual's interactions with the physical and social world. Her more recent work has elaborated these themes in the context of a study of apprenticeship among West African tailors.

Richard Rorty is an American philosopher who, after a long and distinguished career as an analytic philosopher, embraced the Continental philosophies of Derrida and Heidegger and began to question the claim of philosophy to a foundational role and, more particularly, the central claims of the traditional Anglo-American philosophies of mind and language. His book *Philosophy and the Mirror of Nature* (1979) deliberately subverts these projects from within with arguments on their own terms. He also describes in clear terms (in Chapters 1 and 3) the history of mentalism, thus allowing it to be seen as the contingent result of particular choices rather than as an invisibly monolithic fact.

Harvey Sacks was an American sociologist who had an uncannily precise eye for the methods by which people maintain the often-invisible rules of social conduct. Along with Emanuel Schegloff and Gail Jefferson, he founded a discipline of *conversation analysis*, which investigates the properties of ordinary, naturally occurring conversations through extraordinarily detailed studies of tape recordings and videotapes. For example, one of the first important papers of the field, (Sacks, Schegloff, and Jefferson 1978), is a close analysis of the dynamics of turn-taking in conversation. Sacks, unfortunately, died before writing very much. The vast majority of his ideas are only available in transcripts of his lectures. These lectures are currently being edited for publication.

Harry Stack Sullivan was an American psychiatrist whose *interpersonal theory of psychiatry* centered on the interactions ('dynamisms') between people rather than on structures and processes in their heads (1953). His psychology traces the patterns of interaction characteristic of various points in the human life cycle and traces the ways they can go wrong through unfortunately formative influences. Sullivan's work can be refreshing insofar as he was the last important clinical psychologist not to be heavily influenced by Freud. His writing is also perfectly clear.

Lev Vygotsky was a Russian social psychologist who emphasized the role of the social environment in individual development. His principal work was his book *Thought and Language* (1934). (For an anthology of his articles see Vygotsky 1978.) One of

Vygotsky's important ideas is that cognition arises through the internalization of collaborative activity. This idea has inspired a productive school of Soviet *activity theory* that investigates human psychology in its social context. Much of this work has not been translated into English, but an introduction to it is available in (Wertsch 1985).

Donald Winnicott was a British psychoanalyst who brought his long experience as a pediatrician to the task of psychoanalytically reconstructing the earliest experiences of his adult patients. It must be said that psychoanalysis has come a very long way since the crude, sexist, mechanistic ideas that originated with Freud. The credit for these subsequent developments must be split many ways, but Winnicott was an important figure in the early development of the object relations school, which focused on the structure of individual's formative interactions with other people, particularly their mothers, and the consequences of these interactions for one's experience of human interactions later in life. Winnicott emphasized the basic trust in the world that is necessary for healthy living. In particular, he described the conditions that permit the infant to feel able to embrace the reality of a world independent of its own desires. Through investigation of the consequences of an untrustworthy early environment, Winnicott described the kind of early basic emotional contact and practical support he referred to as *holding*. Winnicott's views strongly influenced the theme of dynamic holism that Chapter A3 begins to develop. Winnicott's writing is a model of clarity. Start with his anthology *Playing and Reality* (1971) or with his posthumously edited manuscript *On Human Nature* (1988).

Chapter A3

Walking to the subway

Joshu asked Nansen: "What is the path?"
Nansen said: "Everyday life is the path."
Joshu asked: "Can it be studied?"
Nansen said: "If you try to study, you will be far away from it."
Joshu asked: "If I do not study, how can I know it is the path?"
Nansen said: "The path does not belong to the perception world, neither does it belong to the nonperception world. Cognition is a delusion and noncognition is senseless. If you want to reach the true path beyond doubt, place yourself in the same freedom as the sky. You name it neither good nor not-good."
At these words Joshu was enlightened.

Mumon's comment: Nansen could melt Joshu's frozen doubts at once when Joshu asked his questions. I doubt though if Joshu reached the point that Nansen did. He needed thirty more years of study.

In spring, hundreds of flowers; in autumn, a harvest moon;
In summer, a refreshing breeze; in winter, snow will accompany you.
If useless things do not hang in your mind,
Any season is a good season for you.

Ekai, *The Gateless Gate*, 1228. In Paul Reps, ed, *Zen Flesh, Zen Bones*, Anchor Press, no date.

A3a Context and summary

I live in Boston, in a loft on the top floor of an old factory building at the relatively safe end of Edinboro St, a dirty, noisy side street in Chinatown. On about 400 mornings over three years I have walked from my home to the Washington St subway station, a distance of about three blocks, starting down Edinboro St and crossing Essex St to the Avenue

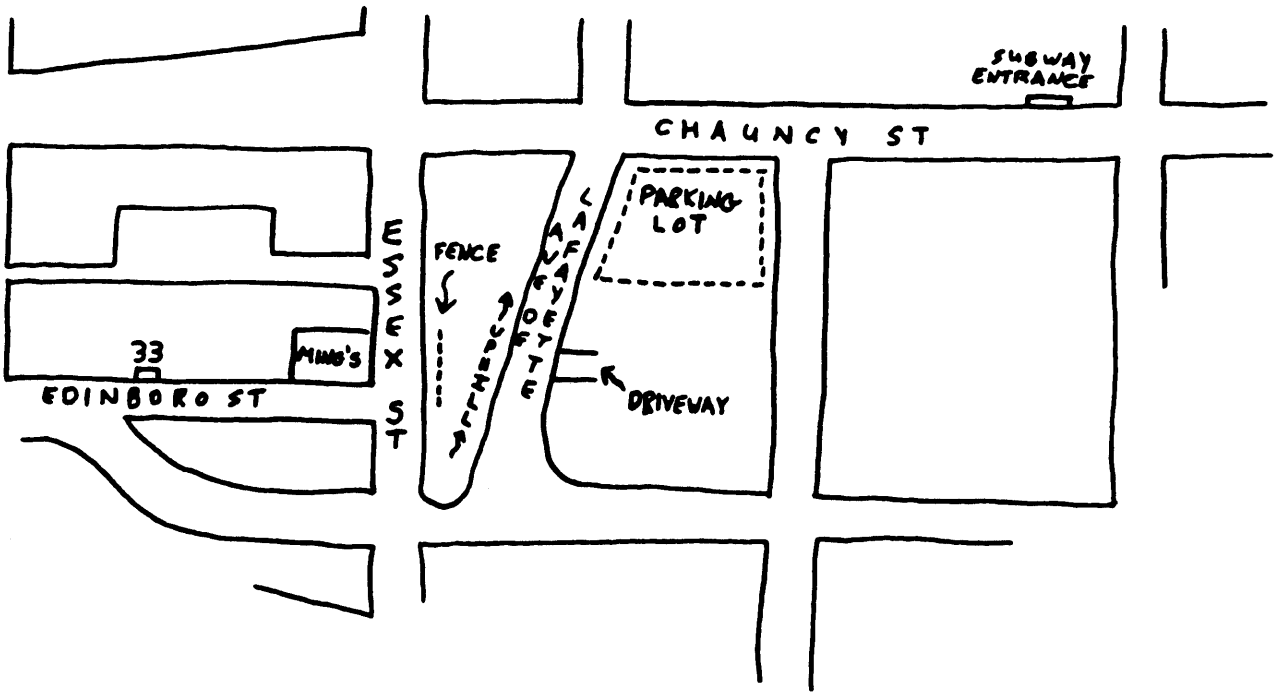

Figure 1. Boston's Chinatown *circa* 1986. I have often walked from my apartment on Edinboro St to the Washington St subway station entrance on Chauncy St.

de Lafayette (no kidding), which I cross to cut diagonally across a parking lot and continue down Chauncy St to the subway entrance. Each morning's navigation of these three blocks is informed by a great deal of history. I propose to narrate my morning's walk to the subway, pausing along the way to point out and summarize some important dynamic themes. (The route evolves continually, both because I discover new aspects of the landscape and because the landscape changes. I wrote the narration in early 1986 and the dynamic analysis in early 1987.) These themes have been omnipresent in my investigations of routine activity. Together with the dynamics of routine activity in workspaces, they are the principal motivation for my theories of machinery.

I should make clear that this narrative is not a 'complete theory' or a 'worked-out example'. One should not expect this to be possible—at least without massive simplification—since most of the important dynamic phenomena occur in any non-trivial activity. To exhaust this example would be to present a complete theory of human activity. (Indeed, this is regularly true of narratives covering half a second.) Interesting dynamic issues are involved in keeping my balance, sighting the spot where I'll put my foot when stepping onto a curb, fishing a subway token out of my pocket, and so on, but I don't understand them very well.

As a summary, here are the important dynamic issues that *will* arise here.

1. The nature of plans. No plan could ever be so exhaustive that you could mechanically 'execute' it. Carrying out a plan requires continual improvisation, in-

terpretation, and fine judgement—especially about whether to revise or abandon the plan in mid-course. This is just as well, given that you're probably not doing anything else with your brain while you're out there following the plan. Real plans can be concise compared to Plans because they can rely on many aspects of how, where, and by whom they'll be used. See (Agre and Chapman 1988) for a longer discussion of plans.

2. Routines and their evolution. Everyday life is, for the most part, routine. Our everyday interactions with familiar people, places, and things tend to fall into recurring patterns called *routines*. This chapter, for example, describes my routine for walking from my apartment to the subway. A routine, like any other dynamic entity, is just a theorist's construct. A routines, in particular, is not a plan. Someone engages in a routine because, for whatever reason, their interactions with a certain environment regularly work out a certain way. Routines are not plotted out from scratch. Rather they evolve as the relevant agents pursue their ordinary activities. New routines can evolve fairly quickly, but the vast majority of everyday routines evolve slowly if at all. Section B2e describes routines and their properties in more detail.

3. The accumulation of responses. When you do something repeatedly, you accumulate a repertoire of methods for dealing with the opportunities and contingencies it tends to present. Individually these methods are useful for saving the trouble of figuring them out again. Collectively they're an efficient substitute for paranoia: after a while you can be confident you've seen it all. You can carry on assuming that you needn't have any worries except the ones that come to mind on their own. Section B2e will discuss further the dynamics of accumulated responses.

4. Visual routines. The dynamics of vision in real activities suggests a resolution of the conflict between the 'bottom-up' and 'top-down' views. The periphery continually performs calculations uniformly over the present pair of images; the center continually applies a massive collection of possible lines of reasoning to the problem of what to do now. At the boundary between them, the center poses a serial stream of queries to the periphery. This view derives from Ullman's notion of *visual routines*. Chapters C3 and C4 will describe visual routines in detail and discuss an implementation of them.

5. Improvisation. Everyday activity, however routine, is not a matter of mechanically following a plan. You might or might not have plans and signs and shopping lists to help you carry on your daily activities, but in any event you must continually redecide what to do. Everyday activity is, in this sense, fundamentally improvisatory. Our activity takes account of multitudinous details of our surroundings, conforming to their particular arrangements and continually either recommitting to a course of action or else choosing a new one. The dynamics of improvisation

will take different forms in different sorts of activities. Chapter C4 will discuss the notion of improvisation in more detail.

Later we will see how some of these dynamics arise through interactions between particular sorts of machinery and particular sorts of worlds.

A3b The route

Walking in the city is not an abstract exercise. There is a city in front of you at all times, making your task both easy and hard in ways that maps omit. City streets are complicated places, crammed with people and things that have different significances for different people at different times. Boston's navigational peculiarities are notorious. We talk about the specificity of a place, but for 95% of the mundane business of walking around, all cities are identical. To start with, sidewalks are paths.

Following a path is an extreme of unplanned activity. Paths tend to dissolve time by presenting themselves as a heap of disconnected events. The landmarks along even a very familiar path can be hard to recall in order unless some logic connects each one to the next. This would matter if towns exchanged places in the night. Paths tell you where to go. They don't need maps.

My route to the subway isn't an unambiguous path, but it still doesn't require much of a plan. I don't remember if I first did it from directions or if I was shown. In any event if I were to direct you to the subway, you wouldn't need any more plan than "left out the door, cross straight over Essex then left up the hill, take the first right and it'll be on your left," which is nothing next to the actual complexity of the trip. Consider how much these directions leave out. "The door" is presumably the front door of the building. There's no need to tell you to walk down Edinboro St in the direction that "left out the door" will leave you headed; when you're on a path you don't need a plan. No mention, either, of the fact that Essex St is not marked as such at its intersection with Edinboro St, nor of the fact that it will be still entirely clear which street was meant once you get there. (Experimental subjects given these directions were actually bothered by a lack of a marking, though they got to the subway without incident.) "Left up the hill" will manage to refer to the Avenue rather than to Essex St because it's the only hill you can see when you're standing at that intersection looking that way. (If you turn around and look hard, you can see another hill near the expressway interchange about half a mile away.) Getting to the Avenue will require a brief rightward detour to get around a fence. No need to mention either this detour or the necessity of crossing the Avenue. The directions leave out the parking lot altogether; presumably you will have the sense to see the first right coming and cut the corner; and it doesn't matter if you don't. You'll also need the sense not to interpret a driveway or the parking lot itself as that first right. Everyone relies heavily on these sorts of things when giving directions. Some people are better at it than others. For example, experienced urban direction-givers know that alleys often confuse people who've been directed to count lefts or rights.

When you're following a plan, your surroundings are available as a resource for interpreting it. The existence of this resource in turn influences the phrasing of the plan.

> I was ill and consulting with a doctor. He wanted me to get an X-ray, so he gave me some forms and told me to walk over to the hospital, several blocks away. After determining that I knew where the hospital was, he said "Go to the X-ray department. I'd explain where it is but it's too complicated. Ask around, you'll find it." (I figured he must have been through this before.) In the event, the only complicated part was locating a suitable entrance door, the sign for which managed to escape me because of the architectural chaos in the section of the hospital grounds I had walked into. There were many doors but all had unpromising labels. At length the door labeled "Neurosciences Outpatient Reception" sounded promising; as I approached it I finally saw the signs labeling it as the entrance door I was seeking. (It was obviously an entrance door once I managed to focus on it.) After walking in, I scanned the opposite wall and immediately spotted a map with a YOU ARE HERE. After scanning the map for about five seconds I found X-RAY, found the YOU ARE HERE again, concluded I should turn right, and doing so, immediately picked up a trail of X-RAY signs with arrows.

Likewise, a plan that refers to "the hill" counts (roughly speaking) on there only being one hill apparent to someone who has gotten that far in the plan. A plan that instructs you to "take the first right" counts on it being clear which street is indicated. 'Counts' and 'clear' are defined reflexively, almost circularly, as that which a given person will be able to figure out in a given situation. In particular, they leave it implicit that one counts streets, not alleys and passageways and subway entrances. There's probably not a rule for determining what to count, but it doesn't matter if it'll be clear when you get there.

The plan also relies on your experience and skill. The instruction to "walk down Edinboro St" assumes you have the sense to disobey it when the street is full of slush or garbage or worrisome people, as it often is. The plan omits things you already know, like how to cross a street, how to use street signs, how to notice another street coming up, and where it's safe and legal to walk. It also omits things you can be trusted to figure out for yourself, like how to recognize the subway station, how to wind your way past the trash strewn outside Ming's grocery, and how to get some new directions if you get lost. In short, plans are abbreviated in each of the many ways that you and I share an understanding of the world (*cf.* Suchman 1987). (Future work will explore the extremely complicated dynamics of these abbreviations. They have been extensively documented by Garfinkel and others.) This is the only reason plans can be written down at all. Plans are not algorithms; algorithms only work inside computers. Programming texts that compare algorithms to recipes severely confuse this critical issue. (For more about the nature of recipes, see Scher 1984. For a typical statement of the view that

recipes are at best defective computer programs see Knuth 1984 page 6. See also Waite 1975.)

A3c Edinboro St

> "One only knows a spot once one has experienced it in as many dimensions as possible. You have to have approached a place from all four cardinal points if you want to take it in, and what's more, you also have to have left it from all these points. Otherwise it will quite unexpectedly cross your path three or four times before you are prepared to discover it. One stage further, and you seek it out, you orient yourself by it. The same thing with houses. It is only after having crept along a series of them in search of a very specific one that you come to learn what they contain. From the arches of gates, on the frames of house doors, in letters of varying sizes, black, blue, yellow, red, in the shape of arrows or in the image of boots or freshly-ironed laundry or a worn stoop or a stairway's solid landing, the life leaps out at you, combative, determined, mute. You have to have traveled the streets by streetcar to realize how this running battle continues up along the various stories and finally reaches its decisive pitch on the roofs. Only the strongest, most venerable slogans or commercial signboards manage to survive at this height and it is only from the air that one can survey the industrial elite of the town ... beneath one's eyes."
>
> Walter Benjamin, *Moscow Diary*, 1926, p. 25.

A path may tell you everything you need to know, but my path to the subway tells me more than it used to. Anticipating a little, let us consider a certain tree along the Avenue de Lafayette. Once "just a tree," this tree became "the tree that hangs too low over the sidewalk," so I started walking around it. This transmutation didn't occasion any fanfare; I'm sure a hundred more like it have escaped my theoretical curiosity and go unremarked to this day. The tree presumably owes its significance to some event from my past that somehow changed me. Perhaps the tree has been assigned a name which figures in full-blown representations relating it to axioms of eye-poking and circumnavigation. But that seems like overkill. The psychic residue of my experience with the tree has exactly one job: to notice the tree approaching and remind me of its low clearance. Except for the very moment I'm rounding the corner onto the Avenue, that bit of mental stuff is best to hide in some dormant lobe. In fact that's what it does; I only managed to exhume it for you because my interest in navigation routines made me notice my aversion to this tree one morning.

To avoid trivializing my case, I should distinguish two kinds of results of the history of this walk. First there are the 'associations' and 'things I'm reminded of'. There are hundreds of bits of text on doors and windows that my eyes happen past; they're all familiar each time but I can only recall a few of them. The sincerely strident graffiti

on various walls reminds me of its author, a remarkable individual late a neighbor of ours. The incongruously unfaded yellow lines on the Avenue de Lafayette reminds me that this historic-sounding boulevard was actually built in 1985 as an access road to a shopping mall. Being reminded of these things no longer provokes any new trains of thought, much less any memory of having been reminded of them. Their only distinction lies in being so phenomenologically peripheral.

Reliable as they are, these effects make for less compelling arguments than experiences that have influenced where and how I walk. The tree example is not particularly consequential because my response to my tree is unvarying and might seem somehow programmed. The calculations I make while walking down Edinboro St are more complex. Though almost always perfectly routine, these calculations vary from moment to moment and day to day as patterns of life evolve and experience accumulates. I have never been assaulted or otherwise damaged on Edinboro St. On the other hand, I have been obstructed by delivery trucks and garbage piles, bumped into by handtrucks, intimidated by suspicious people, buzzed by cars of revellers driving on the sidewalk, and felled by wintertime ice patches. I've also had occasion to step in pothole puddles, ankle-deep slush, and the fishy slime that Ming washes from his loading dock. These things don't happen very often, nor have they left any deep marks on me. Nonetheless, they enter into every morning's walk in a routine, unspectacular way. Every moment's step forward has a collection of precedents that apply themselves with no discernable effort. I now anticipate the sections of curb where puddles collect, the handtrucks emerging from behind the delivery trucks, the icky sheen on Ming's sidewalk, and so on. Long before the activity behind Ming's hold me up I've checked if the sidewalk is passable and crossed if it's not. I'm sure I couldn't write a program to assess the varying configurations of people, delivery trucks, garbage piles, discarded cleaning water, stacked vegetable crates, and large angry fish being transferred between tanks. Such a judgement becomes routine asymptotically as some space of combinations plays itself out. Of course there's no reason to have any sense of this space, much less to imagine what's left of it. On any given morning I deal with what's there. One morning I found a large fish lashing about in my face.

If I never changed, I'd fall for the pothole puddles every morning. Obviously my experiences have changed me. As Part B explains, I would like to explain these changes in terms of dependency maintenance. The accumulation of responses corresponds to an accumulation of circuitry; some combination of existing circuitry operates each morning.

My accumulated responses are useful both individually and as a group. Individually, they save me the trouble of figuring out how to get around the trash, jump the puddle in the gutter, wait for the store workers to go by with their loads, and so on. The first time I had to take these actions, I had to stop, look around, figure out what was going on, consider some options, choose one of them, look for any obvious pitfills, and go. This took some thought, some looking around, and some time. Now I only perform the minimum necessary thinking and acting, and doing so doesn't interrupt anything else.

Considered as a group, one's accumulated responses offer a solution to the frame problem. If you're considering taking some action, there is no way to prove that nothing bad will come of it. Our thought underdetermines the world in endless ways. There is simply not enough information. The frame problem renders any fixed algorithm incompetent to decide what to do next. For example, one cannot address this difficulty by inquiring into the properties of a single unified substance called 'uncertainty'. Calling some mathematical idea by this name only confuses the issue. There are no proofs. Existing planners can only prove the correctness of their plans because their domains are artificially simple and their domain models are artificially tractable. Moreover, a planner *must* prove the correctness of its plans. Otherwise there is no reason to believe the plans will work.

The frame problem requires a holistic solution: not holistic machinery but a particular sort of dynamic holism. As you become experienced at something, you accumulate ways of anticipating difficulties. Each pitfall in your catalog asserts itself whenever it becomes applicable. Any action you consider gets performed unless a candidate problem asserts itself. You're optimistic by default; you count on your accumulated responses to qualify your optimism. This strategy sounds dangerous, but cultures are set up to make it work by compensating for its systematic lapses and offering practices that amplify its effect. One of these practices is the trick of looking around before taking a novel action, just in case you spot something that signals a warning. Every morning, I walk down Edinboro St like nothing's wrong. If I spot a delivery person with a handtruck, I routinely figure I should cross the street to get out of the way. But if no warnings occur to me, I walk straight on.

A3d Crossing Essex St

Using experience requires subtlety; one shouldn't give generalizations too much priority over the actual circumstances of *this* morning's walk. It's all right to keep avoiding the low spots even when it's dry, but specificity matters when you're crossing a busy street. Not having grown up near a city I experience crossing a city street as a skill. What is this skill made of? Most of the skill seems to apply to new streets, but at the same time crossing Essex St seems a skill of its own. Essex St constricts to one narrow lane in front of the Chinese markets as people load groceries into double-parked cars. Drivers who see daylight at the end of this channel have long lost any sympathy for pedestrians. The eye is good at extrapolating accelerating objects in rational situations—thus, trucks accelerate slowly and so leave gaps in traffic—but not here. This particular generalization has probably saved my life.

Once a hole opens in traffic and I start across Essex St, the scale of the landscape suddenly increases as a big hunk of territory opens up. For the first time I actually see the far sidewalk of Essex St, the rise along the Avenue de Lafayette, and the large open space to which it leads. What's most remarkable about this space is the invisibility of the buildings that bound it. Straight ahead is a featureless cube of beige brick I

associate vaguely with the MBTA (the public transit authority). To the far left is the back of a tall dark grey shopping mall–hotel–parking garage (after which the Avenue de Lafayette is named, you see) which only registers as a preconscious absence of sky. To the sides are two 1900-era buildings whose functions, aside from a few street-level shops, are a mystery to me. In the middle is a parking lot marking time until a building appears on it. (Actually, since I wrote this the building has appeared.) The parking lot is invisible too at first because it is about three feet lower than the top of the Avenue de Lafeyette (the only change in elevation, aside from expressway ramps, for several blocks). In Chinatown your body acts as a whole picking its way through traffic, but in these planned urban spaces the eye is in charge, plotting straight lines over each next patch of terrain.

And so even though I'm still in the middle of Essex St my eye is plotting a straight line up to the Avenue de Lafeyette sidewalk. But the joke is on my eye because on the far side of Essex St there is generally a parked car centered precisely on that line. When this happens, I often find myself suspended between two perfectly symmetrical ways of circumventing around the car, left and right, since nothing recommends either direction over the other. The ensuing deadlock reminds me that I am here, now, crossing a busy street by the transient grace of the traffic. This deadlock could be resolved by appeal to astrology for all I care, but because nothing sensible can discriminate I have to focus my thinking for a moment and formulate an arbitrary way of deciding. Everyone who has spent fifteen minutes staring indecisively at a menu knows what I mean.

A3e L'Avenue de Lafayette and the parking lot

Walking in the city requires a continual negotiation between imaginary straight lines and genuine automobiles. If you need to cross a street that's perpendicular to your path, you have to wait for a hole in traffic but then you can keep going on the same course. If you're walking along a street and need to cross eventually, you get to choose the right moment. On quiet streets you can wait for a comfortable gap between the parked cars, something your eye can pick out at a distance. On busier streets you can cross the parked cars first and then walk along in the street waiting for a safe gap among the moving ones. The angle at which you cross a street balances forward progress and safety according to some calculation that resides in the body rather than the mind. The mind is now in charge of crossing the Avenue de Lafayette, however, because the wisdom of the body hands me over to the low-hanging tree I mentioned earlier. One day as I approached the tree I decided to head straight for the near corner of the parking lot rather than crossing the street at a more conventionally acute angle. The Avenue conducts very little traffic for all its landscaping, but even so this little innovation appears to run afoul of the tendency of early training in our culture to treat potential moving cars as actual. So potential cars faintly materialize as a matter of routine and the stray actual car delivers a routine echo of a told-you-so.

Now I'm at the top of the low hill, approaching the near corner of the parking lot.

The lot is at the prevailing elevation, the hill about three feet above. Accordingly, there is an asphalted slope between me and the lot. Once I clear the last tree I can see the whole lot. The ensuing diagonal crossing of the parking lot is the hardest case of the negotiation of straight lines and parked cars. My first job is to pick a direction to walk in. Let us dwell a moment on the images falling on my eyes. They depict stationary and moving cars, pavement in a few different textures, concrete sidewalks, the lot clerk's shed, trees, people, buildings, signs, lamp posts, trash cans, and assorted rubbish. I want to leave on Chauncy St from the opposite corner of the parking lot. To pick out the spot I'm headed for, I should just look straight ahead and find the occluding contour of the building in the center of the image. Unobstructed paths among the cars show up as black strips headed in that general direction. Most likely there will be a large black strip heading for the lot's entrance, somewhat off to the right of my desired direction. Usually there are others.

Your kitchen looks entirely different according to whether you intend to make breakfast or find the cat. Likewise, this same parking lot scene would look entirely different if I was looking for my car. Instead of inspecting the buildings and the black patches and interpreting everything else as obstacles, I'd be picking out blue patches and checking to see if any of them is my car. I pick out my car from among the other ten thousand blue Japanese cars in Boston by looking at its dents, its lack of bumper stickers, the flying carp windsock hanging in its rear right window, and the duct tape over its rust holes—whichever is visible. Thus, the goal I'm pursuing powerfully constrains what I'd like to know next about my visual field. I'd like to pick out a depth discontinuity, or a black strip, or a blue patch, wherever in a large region it might lie.

We might caricature AI vision research as a dispute between two opposed tendencies. The older one emphasized 'top-down' goal-directed querying of the image and paid little regard to the physics underlying the image (cf Winston 1975). A more recent movement has emphasized 'bottom-up' processing of the image by a collection of prewired modules (Marr 1982). You can make either tendency plausible *a priori* if you pick your prototypical examples right. If you inspect a retinal image out of all context, all that seems plausible is to 'recognize' or 'classify' everything in sight and make a complete model of the whole scene. The right way to adjudicate this dispute is to ask what real people care to know about visual images in the course of real activities.

Part C will argue for the view that the parking lot example hints at. The machinery consists of a periphery and a center. The periphery is precisely the sort of bottom-up device envisioned by Marr. The center, by contrast, is constructed during one's development. There is a very clear boundary between the two, almost exactly as described by Ullman (1984). There is a strong sense of focus; a small number of image features can be *marked*. The center poses the periphery tasks like "pick out a blue thing," "pick out a line," and "follow the marked contour." The exact 'instruction set' of these queries is unknown, to be determined both by psychophysics and dynamic analysis. Chapter C4 works out an example.

Let us continue into the parking lot. Sometimes my eye will pick out a convenient

channel through the cars—the eye is very good at this. But usually it doesn't, and my body will complain at the prospect of tacking a Manhattan-style approximation to the theoretical Line from Here to There. The eye, knowing only the Theory of Straight Lines, responds by reassigning There to some new spot on the edge of the lot, generally following the main path to the entrance on the right. Then often a moment later I'll change course again as a more direct channel appears. An agent that understands its activities will modify its subgoals when they become inconvenient. In this parking lot, the management of subgoals is a matter of sighting new courses. In other domains the sense of perspective and proportion required to maintain subgoals is more abstract, but here the eyes are in charge because all the relevant considerations lie in visible geometry.

Some version of this drama of subgoals takes place every morning. There is so much variation in the pattern of parked cars in this lot that I have no fixed policy about how to cross it, but there's a routine nonetheless. The routine lies in the negotiation of subgoals between eyes and body. Nowadays when my body complains my eyes *just* set about finding a more navigable channel, without breaking my stride or distracting me from my distractions. This routine would no doubt simplify itself if there were never a roughly diagonal channel among the cars, or if there were always one. The routine has tuned itself to the actual variation of the parking lot. This tuning is different from compulsive optimization; it is simply the accumulated consequence of my having walked to work on a set of mornings.

A3f Chauncy St and the subway entrance

> "The buried paths of the Boston subway could not be related to the rest of the environment except where they come up for air, as in crossing the river. The surface entrances of the stations may be strategic nodes in the city, but they are related along invisible conceptual linkages. The subway is a disconnected nether world, and it is intriguing to speculate what means might be used to mesh it into the structure of the whole."
>
> Kevin Lynch, *The Image of the City*, MIT Press, 1960, p. 57.
>
> "Most of [the Boston subway stations] are hard to relate structurally to the ground above them, but some are particularly confusing, such as the utter directionlessness of the upper-level station at Washington St."
>
> Lynch, p. 74.

The Chauncy St entrance to the Washington subway station is the back entrance; the front entrance has a number of branches a block away on the intersection of Summer St and Washington St. If I imagine very hard I can picture the train to Cambridge traveling along under Summer St to the Boston Common and beyond.

But for purposes of going to work in the morning, there is the entrance and a winding staircase leading down to the underground 'concourse', an ancient smelly passageway

with subway turnstiles at each end and entrances to shops and department stores along the sides. Once I arrive I often find the near set of turnstiles closed. When this happens, I walk the length of the concourse and enter the subway at the other end. That this concourse travels over the train tracks and under Summer St is about as theoretical to me as quarks conserving color.

Consequently I don't think it ever occurred to me to pass the Chauncy St entrance and walk the same distance in the relatively fresh air of Summer St, nor did it occur to me to wonder if some rule governs the turnstiles being open at the Chauncy St end of the concourse. Not, that is, until one day I was walking to the subway with a friend and he recoiled when I moved to walk in the Chauncy St entrance. He very sensibly prefers to risk losing two minutes walking to the main entrance than risk suffering the concourse. Since then I've decided that I haven't noticed any rule to the Chauncy St turnstiles because there isn't any rule to when I go to work. I hypothesize that the turnstiles open at 11 or 12 on weekdays, but I never remember to check.

A3g About stories

> "We have arranged for ourselves a world in which we can live—by positing bodies, lines, planes, causes and effects, motion and rest, form and content; without these articles of faith nobody could now endure life. But that does not prove them. Life is no argument. The conditions of life might include error."
>
> Friedrich Nietzsche, *The Gay Science*, 1882, p 121. (Vintage edition, translated by Walter Kaufmann, 1974.)

This is a good place to post some cautions about this narrative and the other narratives in this report. First, they are not scientific data, they are stories. Every story is consistent with a million actual events; every event is consistent with a million stories. Obviously, the various stories one could tell about a given event vary in their degree of detail and in which details they report. More importantly, every story has its own emphasis, its own metaphors, its own *angle* on the event it recounts. A story is an imposition. It cannot be otherwise. Reality does not come already carved up into objects and events, connections and trends. These are useful fictions. We invent these things—we 'constitute' them—as we pursue our lives, as we continually make sense of what happens in them. When we forget this, we get in the habit of pretending that the particular story we've just told is the last word: that matter is inherently parceled out into the objects our story names, that the metaphors our story invokes are genuine metaphysical categories, that the world was *already* the way we're pretending it is. In ordinary life you usually only need one story at a time, so it doesn't matter if you think it's the only story there is. But in doing research it matters a great deal. Privileging one ontology kills observation by preventing one from ever seeing anything genuinely new. It is also simply a mistake; formalizing human activity in terms of a single ontology is

a useful exercise, but only because it so rapidly makes evident the fallacy of using that particular ontology to the exclusion of others.

Second, the stories I tell can never compel the interpretations I make of them. Usually I have far more reason for my interpretation than shows up in my narration; past a certain point you just have to have been there. There are no proofs. Again, it cannot be otherwise. These theories and the interpretations they suggest need to be judged as a whole against your own experience. They are uncertain, but their uncertainty is not that of introspective evidence; they are *not* introspection, in the sense of sitting still and trying to peek into your own head, whatever that means. One narrates ordinary routine activity; the narrative simply recounts *what happened.* Nor are my interpretations intended as self-evident, as things anyone could see. Today's interpretations are informed by years of working back and forth between new ways of talking about everyday activity and new theories of machinery.

Third, whereas most of my narratives recount particular episodes, the subway story is a summary of what tends to happen when I walk to the subway. It would be hard to record such a long stretch of my own activity without it being corrupted by my being deliberately aware of it. This corruption would be perfectly valid data in itself; after all, it's a psychological phenomenon like any other. In this instance, though, it would be distracting and there would be too much of it to document. In practice, most of the corruption is not in observing the event but in narrating it. When I'm writing out a routine event, I often spontaneously articulate some aspect of it I've never thought of before. Such retrospection can be illuminating, but it's also misleading. It's important to try to separate these newly articulated aspects from the ones that were part of original experience.

Part B

Running arguments

Chapter B1

Context and summary

Perhaps we're all horrendous kludges. Evolution, after all, had to find the right tradeoffs of the various aspects of intelligence against the various aspects of efficiency, and the result had to be only slightly different from a monkey. For the central system, at least, I believe that few such tradeoffs exist. It turns out that the simplest architecture one could imagine can easily support many aspects of the thinking and acting required for the world of everyday life. My argument for this proposition comes in two parts corresponding to Part B and Part C. Part B presents a program, the running argument system, I wrote to experiment with the idea that most activity is improvisatory and routine. The running argument system does not represent a substantial departure from conventional AI practice. Careful analysis of this program in operation, in Chapter B5, will prepare the way for the more radical departures of Part C.

The central idea of the running argument system is that of *dependencies*. A dependency system has a *source* that occasionally thinks a new thought and a *dependency network* that records each new thought and the reasons for it as a bit of network structure. What 'thoughts' are is the source's problem. The network is a combinational logic circuit which continuously drives all the nodes representing propositions to 1 (IN) or 0 (OUT). Two forms of thought are recognized, each joining *reasons* to a *conclusion*. The simpler, "because I believe X and Y I decided Z" becomes an AND gate joining the two reason to the conclusion. For example, "Socrates is a person and all people are mortal so Socrates must be mortal." The more complex, "because I believe X and have no reason to believe Y I decided I might as well believe Z" becomes an AND-NOT gate joining the positive reason and the negative reason to the conclusion. For example, "Socrates says he's mortal and I have no reason to think he's lying so let's assume he's mortal." The network doesn't interpret the propositions in any way, so it doesn't realize that the first example is an instance of a misguided account of generalization or that the second example is only heuristic. The examples illustrate that explaining the architecture doesn't explain how best to use it.

A dependency network has some important properties. It can be implemented easily and extremely efficiently. By remembering only those aspects of a situation that a given

conclusion actually depended on, it generalizes old thoughts to new situations in which irrelevant and perhaps distracting aspects might differ. Even if there are millions of recorded thoughts, it continually decides which ones to apply and which ones to stop applying. It maintains the consistency of the current set of beliefs, relative of course to the source's standards of consistency. And it participates in the dynamics described by the theory of routines.

The dependency network is an enormous combinational logic circuit whose inputs are signals from the sensory systems and whose outputs are signals to the motor systems. A loop is thus set up between the agent and the world. As the world changes, the inputs change, the changes propagate through the network, and the outputs change. The outputs are constantly driving the agent to take action in the world. This action changes the world (which would probably go on changing even if the agent sat still) and the loop recommences. As Section B4d explains, it helps to think of the system as conducting an argument with itself on each cycle. The reasoning has a vaguely dialectical nature—at any moment the set of IN propositions constitutes a potentially enormous argument that the agent should take the actions it is currently taking. As changes propagate through the network, the subset of IN propositions continually incrementally changes. Relatively uncontroversial propositions change infrequently, whereas propositions describing transient states or short-lived actions come and go quite often. As circumstances change the *argument structure* changes as well, constantly selecting a new set of elements from the vast stock of past thoughts. This is called a *running argument.*

The architecture this chapter describes is implausible in two ways. The first is the dependency network's 'source'. Where do the new thoughts come from? It matters, but it's not our topic here. Here I'm concerned with the steady state of routine activity. In this idealized steady state, the source is nowhere in sight. Enough thoughts have been delivered and recorded to produce some sensible response to every situation that might arise. The result is like Simon's ant, cranking out a stream of actions on a fixed repertoire of thoughts implemented in a bit of clockwork. Part B maintains the convenient image of wise new thoughts arriving from a source and being recorded in a dependency network. Part C likewise neglects the issue by positing a wholly fixed network. Of course people do sometimes think new thoughts. But one mustn't hasten to the conclusion that the 'source' is a homunculus, fully general inference mechanism of the sort so much AI work has sought. All a source must do is provide occasional increments of circuitry in some novel situations. How and when this happens is the subject of the theory of routine evolution, with its emphasis on incremental change and the interpersonal and social context of learning. These are obviously big phenomena about which many valuable things have already been said, but they lie beyond the scope of this report. Investigation of their dynamics will, I am certain, whittle away at the homuncular image of the 'source'.

The second implausibility is the expressive poverty of combinational logic. Chapter B5 works hard to make this poverty seem severe. Much of the burden of Part C is to

demonstrate that the trouble lies primarily in traditional programming and representation techniques. The traditional techniques were designed to produce processes within machines, but what we really want are techniques for producing organized interactions between machines and their worlds.

Combinational logic has three problems, whose solutions are the subject of Part C. First, there is no good way to implement variables. This is the critical issue. Variables figure in our only worked-out accounts of abstraction, namely quantified logical formulae and parameterized procedures. In Section C3b I will argue that these accounts are inappropriate for everyday routine activity, on both epistemic and efficiency grounds. In their place I will describe the beginnings of a proposal based on indexical representations. Such representations do not abstract over classes of things-in-the-world that share certain properties but over classes of situations that share certain relationships between the agent and whatever objects are present.

Second, there is no good way to implement data structures that can be inspected by arbitrary processes. The 'representations' I propose are not representations in anything like the traditional sense. In particular they are not datastructures but rather patches of dependency network that originated as indexical thoughts about particular situations. Chapter C2 discusses the issue.

Third, there is no state. There are certainly dynamics which allow the effect of state, including simple things like making notes to oneself. There is also constantly perceivable state in one's body (including perhaps the peripheral systems). And obviously people remember things. I have little to contribute on this topic as yet. All I insist is that state not be interpreted as a world model. Section C3g returns to the point.

Some advice for the reader.

Chapter B2 is about dependencies. It defines the idea of dependencies from scratch and discusses their properties in detail using cartoon examples. This chapter is primarily intended for readers outside the AI community. Except for Section B2e's discussion of the dependency model of routine evolution, all of its ideas are commonplaces among AI people.

Chapter B3 is about the running argument system. It defines the Life rule language and describes its relationship to the dependency system. Little of the technical detail need be understood to appreciate the central ideas of Part B, so most readers should skip ahead as soon as they get lost or bored.

Chapter B4, which discussion of some of the dynamic issues around running arguments, contains the theoretical meat of Part B. It discusses four topics: the difference between Planning and improvisation, the nature of hierarchy and goals, the notion of argumentation, and the notion of running arguments. It illustrates many of its points with stories from everyday life. More technically oriented readers may become impatient with this chapter's lack of detailed descriptions of implemented programs. Such

readers should resort to skimming, skip ahead to the demonstrations in Chapter B5, and return once they have become sated.

Chapter B5, which demonstrates the running argument system in action, contains the technical meat of Part B. It is approximately self-contained. It carefully defines the system's goals, discusses the way in which it is supposed to exemplify these goals, follows the system through several examples, analyzes its performance in great detail, and then assesses how well it has achieved its goals.

Chapter B2

Dependency maintenance

B2a Context and summary

Dependency maintenance provides a simple way to ensure that an old thought will be reapplied in any new situation where it might be useful. What is to count as 'thoughts' is up to whoever thinks them. Some forms of thought are more sensible than others, of course. As later chapters will argue, in a world of routine there is a great advantage to thinking forms of thought that can be efficiently reapplied. This turns out to be a stiff constraint which, however, is miraculously easy to comply with.

I can best summarize dependencies and their importance by contrast with a different account of thought-saving. You could, upon thinking a good thought, take a snapshot of your whole mental state and file it away. Now suppose you figure out how to flip a pancake while on a camping trip. Then you can't use your pancake-flipping snapshot again unless you're either on a camping trip or willing to hallucinate that you're on a camping trip (*cf.* Minsky's idea of k-lines 1986, Chapter 8). Dependencies only save that part of your mental state that the good thought actually *depends on*—pancakes, spatula, wrist action, but not pine smell, mosquito bites, campstove. That way you can reapply the thought, and automatically, any time the reasons are there, even if it wouldn't otherwise have occurred to you.

Despite its simplicity, dependency maintenance offers simple dynamic accounts of a great many important things. These accounts connect situations on which an agent records dependencies to situations in which it uses them. If we know something about the organization of everyday activity, we can predict that an innovation recorded in the dependency network on such-and-such an occasion will find use on certain other occasions.

This chapter's main purpose is to define the notion of dependency maintenance and explain carefully its most important properties. AI people can skim it since most of the ideas will be commonplaces for them.

B2b Main ideas of dependency maintenance

Dependency maintenance starts from two premises: first, that thinking new thoughts is hard, and second, that if a thought has been useful once it's likely to be useful again. A dependency system has a *source*, which has the difficult job of thinking useful new thoughts. By doing the bookkeeping that's required to reapply the old thoughts when they're needed, the dependency system saves its source from having to do this job over again. The dependency system has a simple interface that makes no presuppositions about what counts as 'thoughts'.

First a blizzard of definitions, then some examples. On any moment the source can hand the dependency system a *conclusion* and a set of *reasons*. The conclusion and the reasons are all *propositions*. There are two kinds of reasons, *positive* and *negative*. Every proposition has, at any given time, a *value* of either IN or OUT, meaning roughly believed or not believed. (If a proposition has an OUT value, that doesn't mean that the agent believes that it's false, it only means that the agent has no reason to believe that it's true.) Given a conclusion and some reasons, the dependency system stores a *justification*, declaring that henceforth the conclusion is to be IN whenever all the positive reasons are IN and all the negative reasons are OUT. (Sometimes justifications are also called 'dependencies'.) A proposition might have several justifications; it is IN if any one of them satisfies this condition and OUT otherwise. A proposition with no justifications at all is called a *premise*—propositions are thus divided into premises and conclusions. The value of a premise might be wired IN. Or it might be determined by some other bit of machinery, like a sensor. If so, it's called an *input*. A proposition (probably not a premise) might also directly drive a commands to the periphery. If so, it's called an *output*. See Figure B2.1.

The entire collection of propositions and justifications is called a *dependency network*. Think of a dependency network as a binary logic circuit (regardless of how it happens to be implemented). Each proposition is a *node* which is 1 if it's IN and 0 if it's OUT. Each justification is an n-input AND-NOT gate joining some reasons to some conclusions. A network is said to be *settled* if all the IN conclusions are justified and *consistent* if there is some assignment of IN and OUT to conclusions that the network can settle to. Whenever a premise changes value or a new justification is added to the network, the dependency system somehow finds a new assignment of IN and OUT to the conclusions that settles the network. Whether settling the network is easy and whether the outcome is unique depends on the network, as we'll see in a moment.

(Hayes invented dependencies (1975). So did Stallman and Sussman, independently and a little later (1977). Doyle (1978, 1979) abstracted the functionality of dependencies to produce the first *Truth Maintenance System*, a name Doyle and most others now regret. These systems have been used principally to direct backtracking in languages that express domain-specific search strategies. An important early analysis of the theoretical problems that motivated the invention of dependency-directed backtracking appears in (Sussman and McDermott 1972). An extensive technology of TMS's has grown up,

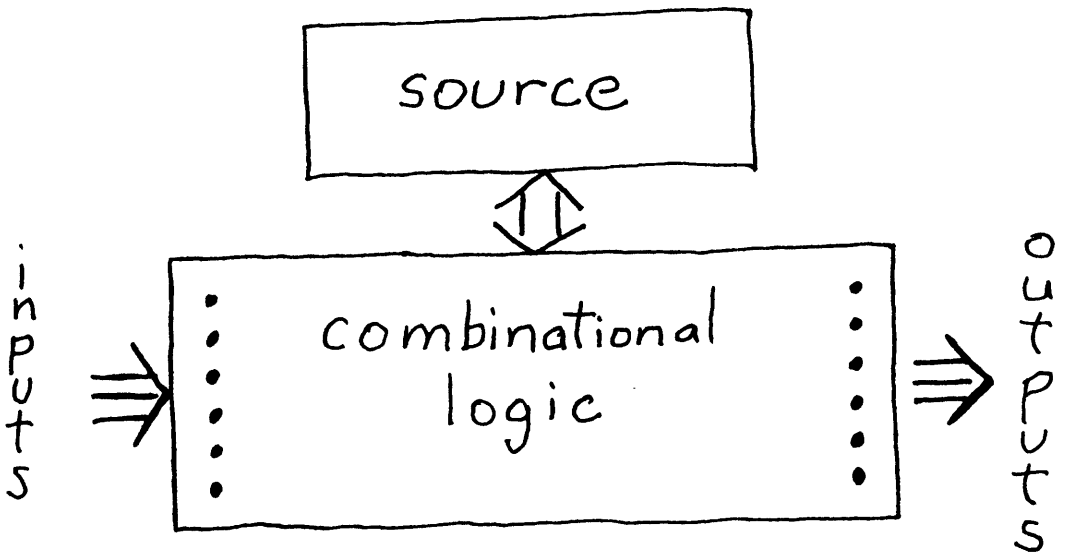

Figure B2.1. A dependency system maintains a dependency network which consists of propositions, modeled as electrical nodes, joined by justifications, modeled as logic gates. Some of the propositions correspond to inputs and others to outputs. The system occasionally adds new circuitry when its 'source' does something new.

including de Kleer's *assumption-based* version, the ATMS (de Kleer 1986, de Kleer and Williams 1987). For an interesting theoretical treatment of the complexity issues that arise in dependency systems, see Provan (forthcoming). Dependencies are similar in spirit to Minsky's idea of *k-lines* (1980; 1986, Chapter 8) and to Carbonell's idea of *derivational analogy* (1983). McAllester has previously used the idea of accumulating lines of reasoning in logic circuits, in his notion of *semantic modulation* and in the lemma library of his proof checker (1988).)

The vocabulary of dependency maintenance is very suggestive, but very few of its connotations are actually intended. Propositions have no internal structure so far as the dependency system is concerned. So far as the dependency system is concerned, they can contain quantifiers, negations, probabilities, or magic words. The reasons might license the conclusion deductively, heuristically, or by divination. If there are new conclusions to be drawn among the existing premises and conclusions, the dependency system will not draw them automatically.

Let us consider some cartoon examples. Notate a justification this way:

```
(<= conclusion
    (in positive-reason1 positive-reason2 ...)
    (out negative-reason1 negative-reason2 ...))
```

(If a justification has no positive reasons I'll omit the empty (in) clause. If it has no negative reasons I'll omit the empty (out) clause.)

To record an ordinary monotonic deduction, use only positive reasons:

```
[Example 1.]
(<= (mortal Socrates)
    (in (for-all x (implies (human x) (mortal x)))
        (human Socrates)))
```

This justification will be implemented by a single 2-input AND gate, joining the two reasons to the conclusion. See Figure B2.2.

For the source, the three propositions in Example 1 have internal structure: it is evidently thinking with first-order logic. The dependency system sees none of this. So far as it's concerned the source said:

```
[Example 2.]
(<= mortal-Socrates
    (in for-all-x-implies-human-x-mortal-x
        human-Socrates))
```

where the propositions have no internal structure at all. For that matter, the dependency system wouldn't care if the source also said:

```
[Example 3.]
(<= (immortal Plato)
    (in (all x (implies (human x) (mortal x)))
        (human Socrates)))
```

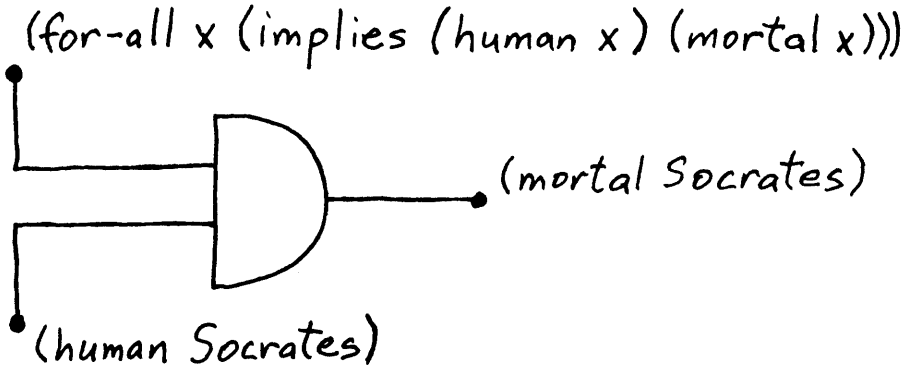

Figure B2.2. A justification with only positive reasons creates on AND gate that assures that its conclusion will be IN whenever both its reasons are IN.

If the source doesn't do such things, our guarantee lies elsewhere. (This chapter will be full of this sort of loose talk about the source. For now, the source works by magic. As the next chapter develops a particular system it will become more precise. The words 'thinking' and 'reasoning', however, will stay sloppy; they refer to whatever action in the machinery I'm discussing in a given context. As words go they're pretty weak.)

A proposition with an empty justification will always be IN:

```
[Example 4.]
(<= (loves Mommy me))
```

The propositions in a dependency network don't have to be examples from logic texts. They are, in fact, more likely to be conclusions about what to do.

```
[Example 5.]
(<= (intend (become philosopher))
    (in (want truth))
    (out (want money)))
```

Just because dependencies are recorded using logic gates doesn't mean the system's reasoning has to be restricted by any particular rules of inference. Heuristic justifications are fair play:

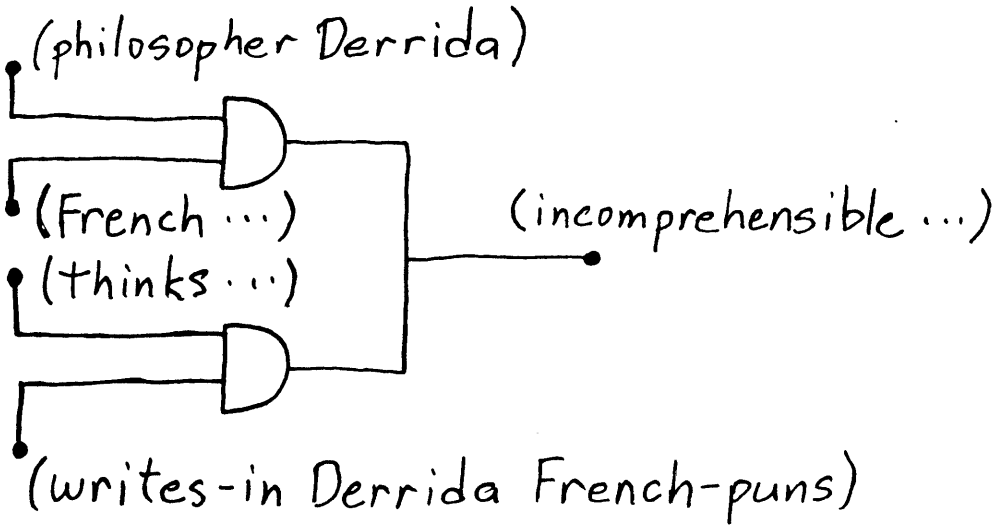

Figure B2.3. When a proposition has more than one justification, any of its justifications can support it.

```
[Example 6.]
(<= (incomprehensible Derrida)
    (in (philosopher Derrida)
        (French Derrida)))
```

A proposition might have several justifications. If the source says:

```
[Example 7.]
(<= (incomprehensible Derrida)
    (in (thinks Derrida (too-easy Heidegger))
        (writes-in Derrida French-puns)))
```

the resulting network will comprise two AND gates with their outputs OR'ed. See Figure B2.3. For compactness, the disjunction is notated as a wired OR.

Let us note in passing that a justification cannot have variables, or better, the dependency system knows nothing of variables. If the source says:

```
[Example 8a.]
(<= (incomprehensible x)
    (in (philosopher x)
        (French x)))
```

then the dependency system will hear:

```
(<= incomprehensible-x
    (in philosopher-x
        French-x))
```

and assign nodes to incomprehensible-x, philosopher-x, and French-x, which probably don't mean anything. If you surmise that several French philosophers are incomprehensible, you have to do it separately for each one:

```
[Example 8b.]
(<= (incomprehensible Barthes)
    (in (philosopher Barthes)
        (French Barthes)))
(<= (incomprehensible Foucault)
    (in (philosopher Foucault)
        (French Foucault)))
(<= (incomprehensible Deleuze)
    (in (philosopher Deleuze)
        (French Deleuze)))
```

Let us refer to a connected subnetwork as a *patch* of the whole network. We see that traditional sorts of representation lead to replicated structure in dependency networks; each French philosopher gets his own patch of network.

Many people have spent much time trying to generalize dependencies to include variables; there seems no way to do it short of reinventing production systems. This is a deep fact. An even deeper fact is that, at least for the sorts of everyday routine activity I have investigated, you don't *need* variables to enjoy the benefits of dependency maintenance. Propositions don't have to refer to objects in the world by names, in the style of first-order logic. Instead, one might say something like:

```
[Example 9.]
(<= (pick-up the-object-I-am-looking-at)
    (in (color the-object-I-am-looking-at gold)
        nobody-looking))
```

This bit of reasoning 'quantifies over' whatever objects you happen to look at. Part C will develop this suggestion.

Negative reasons are a powerful way to express heuristic lines of reasoning. One technique is to establish a default:

```
[Example 10a.]
(<= daytime
    (in at-work)
    (out nighttime))
```

That is, if we're at work then assume it's daytime unless we're sure it's nighttime. This justification is said to be *non-monotonic.* It would be implemented as an AND-NOT gate; see Figure B2.4.

Let's elaborate the example.

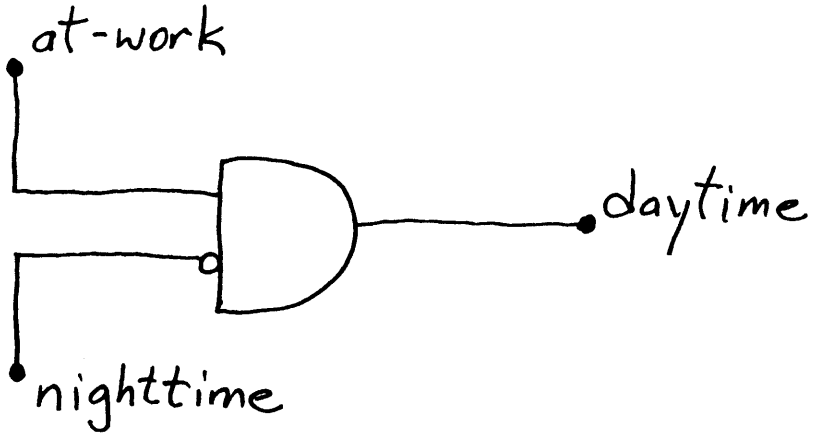

Figure B2.4. Sometimes a justification will have negative reasons, indicating that it should only support its conclusion if all its positive reasons are IN and there is no reason to believe in any of its negative reasons.

```
[Example 10b.]
(<= daytime
    (in out-of-doors
        bright))
(<= nighttime
    (in out-of-doors
        dark))
```

Joining 10a and 10b, we get the network in Figure B2.5. There are six propositions and three justifications. Four of the propositions—*i.e.*, **at-work**, **out-of-doors**, **bright**, and **dark**—are premises because they have no justifications. Let us imagine that the values of **at-work** and **out-of-doors** are determined by other justifications not shown. Let us also imagine that the whole network is located in the head of an agent named Thomas and that **bright** and **dark** are inputs connected to the Thomas' vision system. Their values are continually updated according to how bright it is.

(Remember that these are only cartoon examples. Thinking it's nighttime certainly has more to it than a single wire going high. The point is, whatever compound activity is actually involved will equally well be recorded in the network and reactivated later. The examples will slowly become more real as the chapters go by.)

Now suppose Thomas is at work and indoors. **At-work** is IN and **out-of-doors** is OUT. Probably **bright** will be IN, but since **out-of-doors** is OUT neither **bright** nor **dark** will influence the conclusions at all. **Nighttime** is OUT because **out-of-doors** is OUT. And **daytime** is IN because **at-work** is IN and **nighttime** is OUT.

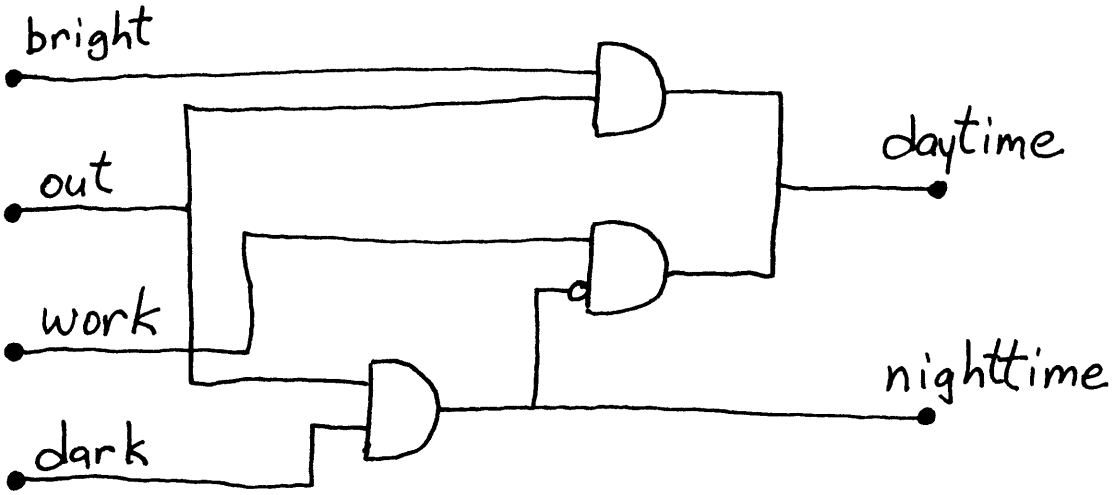

Figure B2.5. In this diagram, the agent will accept certain information as evidence of it being daytime and other evidence of it being nighttime. If no relevant evidence is available, the agent will assume that it is daytime.

Now suppose Thomas gets off work and walks outside. When the clock hits 5:00 **at-work** will go OUT and so **daytime** will go OUT too. Once he gets out the door, it being winter in Boston, **dark** will be IN. Thus **nighttime** will be IN and **daytime** will be OUT.

Thomas's reasoning thus far has some holes. Starting at about 4:30 it'll be dark outside but he, still hard at work, will assume it's daytime. Suppose poor Thomas, longing for a beer as 5:00 approaches, looks out the window where the parking lot usually is and is startled to find it dark. Nobody's perfect, but new circuitry can get you closer.

```
[Example 10c.]
(<= nighttime
    (in looking-out-window
        dark))
```

This new insight, while not spectacular, was no doubt hard work, just because thinking anything new is hard work. But the great thing about dependencies is that they never go away. As soon as tomorrow's approaching beer leads Thomas to look out the window again, this bit of thinking (if you're willing to call it that) will happen automatically. Likewise next week's approaching beers, and next year's. He has permanently gotten that much smarter. With successive new insights, his network will grow larger and larger.

To really get a feel for the dynamics of everyday activity, one should get this event in perspective. Thomas' new thought, even though intended to seem prosaic, was a

fairly unusual accomplishment. If we were to observe him looking out the window on a randomly chosen day, we would see nothing novel, indeed "nothing at all to speak of." (Idioms like this one betray a deep understanding of the real picture latent in the ordinary language. How telling that traditional philosophy has not spoken of them.) Examples like this one are misleading in the way all stories are misleading. Even when it concerns an ostensibly ordinary bloke, a story only peeks in when something exceptionally interesting is getting ready to happen. It chooses and interprets its events through their relevance to a point. It then obscures this imposition by withholding its point until the end, making the interpretations seem natural and presenting the point as flowing from events that were already just-so. Beneath this spurious intelligibility, the actual coherence of everyday activity (which cannot be exhausted by any narrative) flows transparently on. The ordinariness of life goes without saying.

By the way, suppose Thomas also thought something like:

```
[Example 10d.]
(<= nighttime
    (in winter-time
        late-afternoon))
```

It's important to keep in mind that propositions like `late-afternoon` don't get updated by magic. If Thomas hasn't got a sixth sense for wall-clock time, `late-afternoon` will stay OUT, regardless of the time, until some other circuitry drives it IN. But one has many occasions to try guessing a rough time of day, and these ways of guessing will accumulate in one's dependency network, applying themselves whenever they get a chance. Usually one of them will work out.

This brings us to a small matter I haven't mentioned. Not all lapses can be repaired by adding new justifications. A solar eclipse might make it dark in the daytime and a baseball stadium might make it light in the nighttime. But it would have taken impossible foresight to have phrased Example 10b as:

```
[Example 10e.]
(<= daytime
    (in out-of-doors
        bright)
    (out at-baseball-game))
(<= nighttime
    (in out-of-doors
        dark)
    (out solar-eclipse))
```

Any conclusion about a real-life situation is only true *ceteris paribus*. One never stops discovering exceptions to one's general rules. Consequently, the source has to be allowed to add new negative reasons to some of the existing justifications. When Chapter B3 describes how the running argument system uses dependencies, these justifications will

be the ones created by "UNLESS" rules. (This is the only extra feature I haven't told you about.) Adding a new negative reason is called *intervention.*

(Intervention may seem an unmotivated feature right now. The restriction to negative reasons will become clearer when I explain argumentation in Section B4d. Section B2e on the theory of routines describes the role of intervention in the dynamics of routine evolution.)

A dependency network can be circular. This can happen when there are equivalent propositions.

```
[Example 11.]
(<= looking-at-the-Morning-Star
    (in looking-at-the-Evening-Star))
(<= looking-at-the-Evening-Star
    (in looking-at-the-Morning-Star))
```

More generally it can happen when there are several propositions, any n of which imply the rest.

```
[Example 12.]
(<= games=7 (in wins=3 losses=4))
(<= wins=3 (in games=7 losses=4))
(<= losses=4 (in games=7 wins=3))
```

When a dependency network is circular, drawing it as a logic circuit is misleading. I will shortly outlaw circular dependencies.

B2c Dependency networks

Much of this report is concerned with the properties of a dependency network as a whole. A large network might have thousands of nodes, or hundreds of thousands. It might have long chains of justifications. It might have any number of premises. It might be so tangled with circularities that every proposition figures in determining the value of every other. In short, it might be an arbitrary mess. We need words for talking about dependency networks.

Let us define the notion of *support* in a dependency network. Let $P1$ and $P2$ be propositions and let $N1$ and $N2$ be the network nodes that represents them. Informally, proposition $P1$ supports $P2$ if $P1$'s value enters into determining $P2$'s value. The proposition $P2$ might have any number of justifications, each of which might have any number of reasons, both positive and negative. All of those reasons support $P2$ because their values help determine $P2$'s. Support is a transitive relation, so that anything that supports one of $P2$'s reasons supports $P2$. In general, $P1$ supports $P2$ if you can trace a path forwards through the network from $N1$ to $N2$. A proposition's *support set* is the set of its supports. A proposition's *premise set* is the set of premises in its support set.

I've already mentioned *circular* dependency networks. Formally, a network is circular if any proposition supports itself. Circular dependency networks can have some peculiar properties. Some have more than one consistent assignment of IN's and OUT's to nodes for a given set of assignments to premises.

```
[Example 13.]
(<= at-work (out at-home))
(<= at-home (out at-work))
```

A network consisting entirely of these two justifications, while consistent, doesn't contain enough information to pin down a unique assignment. Either **at-work** is IN and **at-home** is OUT or vice versa. The dependency system can choose either assignment arbitrarily.

A circular dependency network can also be inconsistent, meaning that there exist no legal assignments of IN and OUT to its nodes.

```
[Example 14.]
(<= A (in B))
(<= B (out A))
```

If **A** is OUT then **B** is IN, but then **A** must be IN. If **A** is IN then **B**, being unsupported, is OUT, but then **A**, being unsupported, must be OUT.

It makes a big difference whether a network is circular. Every noncircular network is consistent. There is a trivial algorithm for settling noncircular networks because their support relationship is a partial order. This is the algorithm suggested by the logic-gate notation: a noncircular network is a combinational circuit. Every gate continually assigns its output the value determined by its inputs. If a premise's assignment is changed, the change will propagate through the network until it has settled. This will take time proportional to the depth of the network (on appropriate parallel hardware). This method has the tremendous advantage of locality: each gate knows everything it needs to correctly assign a value to its output.

With a circular network, the story is entirely different. The local algorithm, in particular, is not correct. Recall Example 12:

```
(<= games=7 (in wins=3 losses=4))
(<= wins=3 (in games=7 losses=4))
(<= losses=4 (in games=7 wins=3))
```

Suppose we give **wins=3** and **losses=4** their own justifications:

```
(<= wins=3 (in Sox-in-Series))
(<= losses=4 (in Mets-in-Series))
```

In the sad year of 1986, these conditions are all satisfied and **wins=3** and **losses=4** are both IN. Consequently **games=7** is IN, no problem. Next year, **Mets-in-Series** will go OUT, but **losses=4** will stay IN because both **games=7** and **wins=3** will be IN! This may be a good guess, but it certainly isn't fair.

One cannot reliably settle a circular dependency network in time proportional to its depth. Indeed, the problem is NP-complete (McAllester, personal communication). This is a good reason to try to make noncircular dependency networks suffice. I have always found that they do and will simply assume so.

We have now enumerated everything that can happen in a dependency network: gates propagate binary values and new justifications and connections are made. Aside from these operations, the network is *uninspectable*. No algorithm can poke through it, whether to count it, rearrange it, summarize it, decide whether it's circular, compute a proposition's support, or prove that some output will never change its value. It is a network in the hardware sense, not the software sense. Even if it is implemented by an interlinked datastructure in simulation, the accompanying algorithms may only perform the prescribed operations on it. The presumed uninspectability of the dependencies has many consequences, as we'll see in Chapter C2. For example, it becomes hard to implement the common view of representations as inspectable datastructures.

(Intervention through adding a new negative justification is the closest that dependency networks come to being inspectable. Intervention, though, doesn't require anyone to be able to traverse the network and the source knows ahead of time which gates it might want to add negative justifications to under which conditions. I don't know how this works in anyone's brain, though I find recruitment schemes such as those of Marr (1970) and Feldman (1982) appealing. Chapter B3 explains how intervention works in my system.)

We might proceed to understand dependency networks by formalizing their properties in general, proving theorems that apply to any network, no matter how messy and tangled. This is a useful thing to do (*cf.* Provan forthcoming), but it is not what we're doing here. Instead let's ask two dynamic questions:

> What sorts of dependency networks lead their owners into useful sorts of interactions with the world?
>
> What sorts of dependency networks tend to arise in the course of ordinary everyday activity?

We should hope the answers to these questions overlap. They do, as we'll see.

B2d Dependencies as a simple account of many things

Dependency maintenance is a fabulous idea because it provides simple accounts of a great many important things. Let us list these, and in so doing anticipate and summarize the much longer discussions of later chapters.

Suppose we imagine the network to be implemented as actual logic gates or in some other suitably parallel, distributed hardware. The resulting machinery will be simple and easy to construct. It will be blindingly fast, even if the gates themselves are slow. It will apply an enormous amount of reasoning to every moment's activity.

Being fast and automatic, dependencies offer a simple account of how it is you become faster and more fluent at an activity you've practiced. With time, all the necessary thinking has gotten itself set up in your dependency network. When it's needed it just happens. People in AI often speak of this sort of thing using metaphors of 'compilation'. Like a good compiler, dependency maintenance finds opportunities for parallelism in the reasoning it compiles. Two nodes have to be updated in order just in case one supports the other; otherwise everything is updated in parallel. It is best not to take the metaphor of compilation too far. Unlike ordinary compilers, a dependency system doesn't have to manipulate explicit representations of ordering constraints. The 'optimization', like most dynamic phenomena, is epiphenomenal. Furthermore, there is no 'target language' that is interpreted by a stored-program computer. 'Silicon compilation' might be a better metaphor; somebody should try it for real.

In accumulating a history of their owner's past reasoning, dependencies offer a simple account of search control. In AI it has been common to consider Problem Solving or Planning tasks in isolation. Some module must produce a substantial hunk of novel thought in one shot without the benefit of past experience or the possibility of inspecting the world. No wonder it gets itself into uncontrollable searches. Suppose the machinery were to maintain dependencies. All the reasoning that went into past Solutions or Plans would automatically apply itself to the present one. Viewed as search, the dependency network would quickly search those parts of the search space that have been searched before. If this isn't enough, the agent can do whatever the culture teaches one to do: try things, get help, use textbook methods, apply heuristics, take a different tack, cut corners, or walk away. What does this story amount to technically? How sophisticated must one's learning machinery be in a normal culture and world? Must one's architecture provide facilities for search after all? To answer these dynamic questions we must understand what everyday life is like.

Dependencies also provide simple accounts of aspects of reasoning and learning that've generally been thought to require classification hierarchies, pattern databases and matchers, symbolic generalization algorithms, and so forth. All of them follow the same simple slogan:

> There are fewer reasons than causes.

Dependencies record the reasons for thoughts, not the causes. For an example, let's return to Example 10d.

```
(<= nighttime
    (in winter-time
        late-afternoon))
```

Recall that our cartoon agent Thomas, longing for an after-work beer, looked out the window toward the parking lot and found only blackness. This surprise led him to ask someone, or consult an almanac, or perform a correlation, and arrive at a sensible set of

reasons for it being nighttime. The dependency network recorded the reasons (wintertime, late afternoon) but not the causes (weariness, thirst, looking out the window). So this coming Saturday, when Thomas is pulling on his sweats to go out running before supper, he will, without fanfare, recall that it's winter and thus nighttime. He already had sense enough to put on his nighttime running gear at night

```
(<= (put-on nighttime-running-gear)
    (in nighttime
        (putting-on running-gear)))
```

but today he'll think to do it before he gets out on the road. Before, as he was putting on his clothes he had had reasons to dress for nighttime but no causes: there are no obvious clues inside that it's nighttime and nothing led him to notice. But now, a small insight Thomas had in the course of conducting one activity (peering out at his car) has transferred itself to another activity (dressing to go running).

The dynamics by which dependencies transfer novel thoughts to new situations are called the *transfer dynamics.* Here is an example of the transfer dynamics in action.

> I own a dwindling set of wine glasses, sturdy and unfancy ones with fairly thick cylindrical stems. I have evolved a routine for washing them that involves twirling the stem between my right thumb and 2nd and 3rd fingers to run water on it or rinse off the soap, holding it about 45 degrees off vertical, toward the left and away from me, this angle determined by what's both comfortable and visible. I had left one of these wine glasses sitting on the corner of my desk in the course of last Friday's dinner. I noticed this glass at some point and thought I should return it to the kitchen, but didn't do it because I wasn't really headed that way and didn't feel like making a special trip. Finally on the morning in question I was getting the life of my apartment put to bed on my way out the door to work and I saw this glass on my desk so I picked it up and headed for the kitchen. As I had entered the kitchen and was halfway to the sink, I realized that around the moment of the sink's coming into view I had placed the glass in my right hand in the correct grip to execute this twirling motion.

In this story, I believe that an element of my wine-glass-washing routine, namely grasping the stem just so, has transferred itself to a new situation. The premises behind the transfer must have had something to do with holding the glass and an intention to wash it. Evidently one of the premises was predicated on the actual sight of the sink. When I laid eyes on the sink, enough of the premises came in to suggest grasping the glass by its stem. The transfer dynamics have an important role in the evolution of routine patterns of activity. See Section B2e for a brief discussion of the dependency model of routine evolution.

(Incidentally, I am certain that other forms of interaction with my environment could have led me to grasp the stem before I caught sight of the sink. Some of these might even

have involved some sort of 'internal' activity involving visualization or subvocalization. If these happened they would also need to be explained, of course, but in the event they did not. For a broader discussion of some related phenomena concerning 'reminding' and its relationship to memory organization see (Schank 1982, Kolodner and Cullingford 1986).)

Though this report is concerned with steady-state dynamics, future work should characterize the transfer dynamics in more detail. Two important questions arise. (1) How broadly do newly recorded thoughts transfer? This question concerns matters of representation. (2) When you encounter a everyday situation that ought to be routine, does it tend to fall in the scope of a transfer dynamic? Put another way, when are there sufficient causes for a new idea and when are there merely sufficient reasons? On a small scale this question concerns the dynamic structure of everyday life. On a large scale it concerns the dynamic structure of the life cycle in a given culture.

Evidently dependency maintenance gives rise to a sort of automatic generalization. As generalization mechanisms go it is extremely simple. It doesn't inspect any histories of its reasoning. It doesn't crawl up and down any classification hierarchies. It doesn't even substitute any variables for constants. Does the human architecture employ these features? This is an empirical question, and an interesting one, but barring a neurophysiological miracle there's no use in asking it straightaway. First let's ask, does the simpler machinery suffice to engage in the dynamics of ordinary everyday activity?

In the AI literature, the case of Thomas using his understanding of why it's nighttime would be called 'explanation-based generalization' (EBG). A closely related idea is called 'case-based learning' (Kolodner 1986, Hammond 1986). EBG is a simple, powerful idea. To explore it, let's consider another example. As Thomas was out running it was snowing and he came across a rectangular region of pavement with no snow on it. This struck him as odd. Suppose then he came up with an explanation of why this is—never mind how. Maybe he saw a car drive away and leave a similar blank spot. Maybe the spot reminded him of a shadow, so he looked around and noticed cars throwing such shadows. Maybe he asked someone. (A harder question is, why did that blank spot strike him as needing an explanation? Things that need an explanation do not always strike you as such.)

Having produced this explanation in one situation, he can use it in other situations he encountered. At a minimum, he should be able to interpret his next snowless rectangle as a place where a car recently was. He shouldn't have any problem generalizing to motorcycles and trucks. He might also be able to interpret the snowless circles left by trash cans—he should at least have the sense to ask himself what round thing might have been there. How does he do this?

Neither dependency maintenance nor EBG explains where explanations come from; they only explain how explanations lead to generalization. The account offered by dependency maintenance is simple. Your action of adducing reasons in the first case is recorded in the network. When those reasons become true in the second case the explanation will reassert itself. Over your life you've accumulated thousands of such

explanations, all ready to interpret a new situation on a moment's notice. Most EBG schemes are more complicated than this, employing a wide range of searching, substituting, indexing, and categorizing machinery (Mitchell 1983, Mooney and deJong 1985). Certainly these complex methods are more powerful, especially when a poorly designed representation scheme defeats the simple transfer dynamics. On the other hand, they often produce difficult-to-control searches. Are these extra increments of functionality any use in the long run? This is a dynamic question, and a hard one, so you can't answer it without a good dynamic theory. And if you can't answer this question you have no choice but to make your machinery as sophisticated as possible. An easy idea becomes hard. Lacking a complete account of the machinery and dynamics of routine evolution, it is not yet possible to resolve the question.

Generalization is probably the wrong way to think about what dependencies do anyway. Dependencies don't work by manipulating datastructures and constructing a generalized 'idea' or 'proposition' (in the usual sense). They simply make something happen on Tuesday that happened in a more complicated way on Monday. They get the *effect* of generalization, or most of it, without all the machinery and effort.

Dependency maintenance also provides a simple account of belief-system consistency: once the network settles, there are no unjustified IN propositions. This sort of consistency, of course, is only the same as logical consistency in the unlikely event that the source is logical. (Even then it requires the source to detect all its own inconsistencies.) This is just as well; logical consistency is a terribly fragile thing. A better understanding of the interactions between personality structure and the dynamic structure of one's life as whole should help us formulate a more useful notion of personal coherence.

Further, when premises change in a dependency network, the network finds a new consistent assignment incrementally. The propagation of values only affect the parts of the network that need to change, the reasoning they encode being either newly valid or newly invalid. When Thomas looks out the window and finds it dark, he is probably not led to reconsider his job or his politics or how he is holding his pen. In this way, Thomas' dependency network executes a fresh incremental update on every moment. He constantly reconsiders his actions because his perceptual premises constantly change. Chapter B4 discusses this effect, which is called a *running argument*, in more detail. Presented with each next change, Thomas reconsiders only what is relevant and everything that is relevant; what is relevant is determined by what he has found relevant in the past. If he is lucky enough to have encoded all possible relevances in his network then he will never make a mistake. Of course, there is no end to relevance, so Thomas will occasionally fail to make a connection. He might, for example, offer a beer to someone he knows is trying to stop drinking. We all do this sort of thing.

When one's dependency network transfers some reasoning from one part of life to another, it is tempting to call it 'reasoning by analogy'. The usual account of analogical reasoning in AI starts from structural mappings between descriptions (Winston 1979, Gentner 1983). Dependencies, knowing nothing of structures, will not perform such

mappings. Instead, a thought will be transferred between two contexts just in case it can't distinguish them. As with transfer dynamics generally, when and how often this happens depends on both representation issues and dynamic issues. In Chapter C3 I will conjecture that it happens quite a lot because of the indexical and functional properties of our representations. Now, there are certainly times when people deliberately set out to make mappings between representations. Dynamically speaking, this is an exceedingly complicated, culturally specific activity, not to mention difficult and an acquired skill (Gentner and Toupin 1986), that is only required when the simple transfer dynamics don't suffice. We don't tend to notice the simpler cases because they just happen, fluently and efficiently. I expect that two-factor dynamic theories like this one will become common.

Above all, dependency maintenance provides an account of learning through experience. Experience accumulates as a side-effect of purposive activity in ongoing concrete situations and so do dependencies. Whether dependencies are a *sufficient* account of experience is, again, a dynamic question. It's the last question to answer because it must relate an account of everyday activity as a whole to an account of development as a whole. It helps to pose it negatively: how often in the course of ordinary routine activity do you have to think something new? Thinking something new is hard work, requiring an especially auspicious arrangement of circumstance. Five times a second might be OK, five hundred is definitely not. In an activity that's really completely routine, the various transfer dynamics will work together smoothly to deliver the right moves at the right times, recombining things you've learned on past occasions without needing any new thinking at all. Does the proposed machinery satisfy this condition in our world?

The question gets complicated quickly. What is the normal run of variation in ordinary activity like? Can one expect to encounter all the cases asymptotically? How do we even individuate the 'cases'? Above all, how do you know that you won't, suddenly, out of nowhere, in the midst of otherwise ordinary routine carrying on, find yourself high and dry, surrounded by strange creatures and meaningless objects, with no idea how to proceed?

Dependencies are for creatures who live routine lives. In my view, life is routine, and when it's not routine it's almost routine. Almost everything you do is something you've done before. What does this mean? Let us speak strongly for a moment to put things in their proper perspective. Our 'intuitions' about everyday activity only repeat a socially constituted official veneer. Beneath this pretention, the real work is done without any hoopla by dynamic effects that we don't have words for because they don't need names. Proximally and for the most part, our activity is directed at the ongoing concrete situation—conducted in the present tense—even when we think we're 'reasoning' and 'theorizing' and 'abstracting'. Ordinary activity is not like science, nor is it a matter of detachedly poking at the world in order to formulate objective, eternal, predictive, generalized facts about it. We learn from experience not because we try to but because our machinery keeps dependencies. Our old thoughts get reapplied in

the present not because we've generalized them but because they can't distinguish the present from the past (why should they?). If we tell each other we have 'knowledge', discover 'concepts', and make 'plans', those are just clumsy tricks for bringing about a few extra dynamic effects that don't happen naturally. Philosophical poetic justice (in the form of intractable complexity) will be visited upon anyone who tries to base AI on these scare-quoted ideas.

In summary: properly implemented, dependency networks are blindingly fast, massively parallel, and easy to construct. They provide simple accounts of 'compilation', generalization (especially explanation-based generalization), recombining things one has learned on past occasions, reasoning by analogy, belief-system consistency, and acting on accumulated experience. The big job is to demonstrate that these simple accounts suffice.

B2e About routines

My way of walking to the subway, described in Chapter A3, is a routine. The dynamic phenomena of routines and their evolution were my most important motivation in developing the ideas in this report. Time and space have kept me from presenting a detailed account of these dynamics here, but it will be helpful to define the terms, provide some examples, and discuss the dependency model of routine evolution.

By a routine I mean nothing more precise than the vernacular use of the word, as in "my morning routine." In other words, a routine is a frequently repeated pattern of interaction between an agent and the world. A routine is a dynamic. The difference between the words "routine" and "dynamic" is that a routine involves a particular individual (it is, for example, "my routine" or "your routine") whereas a given dynamic might occur in the lives of many individuals.

Here are some typical routines one might engage in: picking up a fork, making a bowl of cereal, measuring out two cups of flour, putting on a heavy backpack, selecting a toll booth at the San Francisco Bay Bridge, putting your watch on, breaking an egg into a bowl, washing the dishes in your kitchen after a large dinner, tossing a wad of paper into the trash can in the far corner of your office, and writing the word "the." It is common to speak of a routine 'for' some task, but a routine is defined by what happens rather than by any endpoint or overall intention. Also, there needn't be any set routine you engage in 'for' any given activity if, for whatever reason, there is no set pattern to the way you engage in that activity. One might specify the task, the circumstances, and the activity constituting the routine in a specific or general way. I might not have any fixed way of pouring liquids in general, but in my morning stupor I might have a set way of pouring my first cup of coffee.

Insofar as a routine is a dynamic, routines have all of the properties of dynamics that Section A2c has enumerated. A routine, like any dynamic phenomenon, is purely a descriptive construct, not a thing in the head, not a plan or procedure. No specific knowledge or competence is required to engage in routines, nor is there a 'routines

module'. If a script (Schank and Abelson 1975 and 1977, Schank 1982) is a mental entity then a routine is not a script, though a script might be considered a *representation* of a routine.

Doing something the same way every time need not result from a specific intention to do it the same way every time. You can engage in routines without in any sense knowing it. In general a routine might involve a series of actions, each a response to the situation resulting from the previous action, without a specific prior intention to perform *that* series of actions. You might weave down your street in the same pattern to avoid the same set of potholes every day just because the potholes are always there. Conceivably you might avoid the potholes in a consistent way solely for the sake of consistency, but more likely your principal concern will be for your car's suspension.

The actions comprising a routine are not dictated by the routine; they are the individual's chosen actions in particular situations. For example, a routine might result from your always improvising the same response to a given situation—perhaps your response is the only sensible one. You might switch your umbrella from your right hand to your left hand so you can use your right hand to get your house keys out of your pocket every single day without ever having made a deliberate policy of it. Furthermore, a routine is not a law of nature; you might have poured your morning coffee the same way a thousand mornings straight, but tomorrow morning your routine might be altered by any of an endless variety of contingencies, from a ringing telephone to a worn-out coffee pot to a spontaneous mystical experience to the onset of a well-earned ulcer.

Different individuals might engage in different routines for the same task. Not everyone has a routine for every task they carry out frequently. It happens that I have rather a strict routine for making an omelette in my kitchen. This routine varies so little not because I am deliberately unvarying but because my kitchen is relatively orderly and because I have put some effort into understanding each of the various aspects of omelette-making. The opinions I've developed about omelette-making guide my actions. But thinking things through isn't a necessary condition for an activity becoming routine. It would *really* need explaining if someone made a series of omelettes and did it differently every time. I can think of five ways this might happen:

(1) They haven't made many omelettes before and they're still learning and screwing it up.
(2) They like to experiment and are deliberately exploring all the different ways of making omelettes.
(3) They are perversely thinking up gratuitous variations every time.
(4) They are often distracted by extraneous matters. Perhaps they don't care much about making omelettes, have other concurrent obligations (like minding a two-year-old), or are persecuted by interferences (like ringing telephones).
(5) For some reason they are always making omelettes in unfamiliar circumstances.

In short, the existence of routines requires no more explanation than determinism. Put the exact same creature in the exact same situation twice and the exact same things

will happen. An appeal to determinism, of course, is only a heuristic explanation. Your coffee-pouring routine doesn't come off 'the same way' every morning down to the last atom. Explaining exactly why routines exist is actually a difficult project that must await a fuller exposition.

Everyday life is made of routines at all scales. Your routine of driving to work has a hundred smaller routines as parts—buckling up, signalling a left turn, picking out the word "police", keeping distance behind the car in front of you—many of which are components of other routines as well. Even the most original and unpredictably improvised activity will be composed of already-routine parts. Aside from being practically inevitable, this property of routines is a computational necessity: nobody could figure out novel forms of activity on all levels of organization at once.

In observing the remarkable complexity of any given episode of real activity, no matter how small, it helps to think of an agent's actions as the result of a long process of routine evolution. The complex forms of interaction do not arise all at once. A new routine might arise in the course of ordinary activity, but then it evolves to more complex forms. Just as one can engage in routines without knowing it, one's routines can—and regularly do—evolve without one knowing it. Most of this undeliberate evolution takes the form of a series of discrete mutations to the routine. You may drive to work one way up until Monday and then slightly differently starting Tuesday.

An evolving routine will tend to take account of more and more detailed aspects of the environment. Actions that were performed serially will begin occurring in parallel. Warning signs become noticed as if expected and precautions are taken without missing a beat. Sometimes a routine will develop divergent lines of evolution in response to variations of circumstance, whether new or previously ignored. Sometimes a routine's evolution will stall in some comfortable pattern, only to resume at some provocation or chance discovery. Bits of action that began as improvisations or afterthoughts become institutionalized, and the boundaries among artificial stages of activity (like preparation, execution, and cleaning up) fade as actions are rearranged and recombined. Workplaces and dwellings begin bearing the marks of one's routines, through both accumulated side-effects and deliberate conventions, and the marks left by each routine prod mutations of the others.

I have observed several kinds of routine mutation. The simpler ones all appear to be instances of the transfer dynamics and the more complex ones all involve patterns of transfer that are induced, even orchestrated, by culturally specific practices for the use of symbolic representations, whether spoken, subvocalized, or written. This is a very substantial claim and I will not pretend to adequately state or defend it here.

Dependency maintenance helps in thinking about the role of the transfer dynamics in routine evolution. In general a routine mutation does not result from a deliberate adoption of a policy or an explicit change to a plan. Instead, the mutation is the result of the agent, faced with a slightly different situation or some new information, having decided on a different action. Having made the novel decision on Tuesday, the agent's dependency machinery will store its reasons and conclusion. Thus the conclusion will be

reasserted whenever those same reasons hold in the future. As a result, *ceteris paribus*, the change to Tuesday's routine will become permanent, at least until some future mutation reverses it, further modifies it, or changes the routine so that the situation never comes up again. Note that the agent benefits from this transfer effect without having to explicitly reflect on the similarities between Tuesday's situation and Monday's. (For evidence on the tendency of routine mutations to become permanent and on several other relevant phenomena, see Neches 1981.) This is the *dependency model of routine evolution.*

The most common form of routine mutation is *backward transfer.* Suppose you engage in a routine that takes a minute. On Monday, at $t = 30sec$ into the routine, you might make a novel observation, perhaps because something went wrong. The observation might not have any consequences on Monday but the dependency machinery records it anyway, connecting its reasons to its conclusion. On Tuesday, those reasons might come IN again at $t = 30sec$ into the routine. More likely, though, they will come IN earlier, perhaps at $t = 15sec$. This effect is an instance of the slogan that novel ideas have fewer reasons than causes. All the causes required to formulate the idea didn't co-occur until $t = 30sec$ on Monday, but the reasons first co-occured at $t = 15sec$. As a result of backward transfer, everything in a routine tends to back up toward the front.

In a cyclical activity, backward transfer will tend to move things countercyclically. Earlier on in a cycle you will know things that you originally noticed later on in the cycle. This will often cause the cycle to change. These two stories illustrate the effect:

> I had a stack of records propped up against a box and I was alphabetizing it according to the artist's name, forming a second, sorted stack propped up next to the first. I would take a record from the top of the first stack with my left hand, find and hold open the right place for it in the second stack with my right hand, place the record in its space, let the stack close over it, and repeat the cycle. After a while I found I was doing something different: whereas before my eyes stayed on the new record until I had picked it up, now I would read the artist's name as soon as I was done with the last record. Then as I picked it up with my left hand, my eyes were already helping my right hand find the right place in the second stack.
>
> I was trying to get a long C program to compile. I was working on a Sun and had divided the screen between two Unix shell windows so I wouldn't have to exit and reenter the editor (GNU Emacs) to run the compiler. I'd run the compiler and it'd get errors, e.g., "syntax error near { on line 173", so I'd go back to the editor window. The only way I knew to get to line 173 was to go to the top of the buffer and go down 172 lines. This got to be a cycle, fixing errors and recompiling. After a while, I found that I would move the editor to the top line before the compiler had even starting generating error messages. (Finally one time the compiler completed without errors and half of me had to skid to a confused halt.)

In each case, fragments of the routine (reading artist's name, moving eyes to second record stack, moving the cursor to the top of the editor) drifted to an earlier point in the cycle. As the routine proceeded, each fragment acquired its own dependency records which explained the reasons for each. As a result, the various actions took place as soon as they could instead of in the more logical order in which they first assembled themselves as I improvised my way through my dealings with each record or syntax error.

B2f Objections

One encounters a number of objections to the idea of dependencies playing a central role in our cognitive machinery.

> Dependency networks, being digital, cannot reason with continuous quantities like size, weight, distance, and loudness.

Dependencies, recall, are only an account of the central system. The sensori-motor machinery of the periphery will almost certainly be largely analog. If we need, for example, to decide which of two weights is greater, we might imagine this to be a peripheral function. Perhaps, then, analog processing is confined to the periphery. Until we settle the empirical issue, we can try to design peripheral faculties that provide enough such functions to comfortably accommodate ordinary activities.

> Justifications, being digital, either apply or don't apply, and thus fail to capture our holistic intuitions about the similarity of non-identical situations.

If there is a single, unified notion of similarity then dependencies are certainly a poor way to compute it. You can make 'similarity' sound as holistic as you like by accumulating 'intuitions' of similarity in a wide variety of situations and observing that they haven't got much in common. But asking what similarity amounts to in particular situations suggests a different picture. At a minimum, similarity depends on what you're trying to do. You might call two chess boards similar if you're playing chess but not if you're considering which chess board to buy. Further, if you're playing chess, then two similar positions are similar for certain reasons. They might both turn on the role of the knights' pawns in maintaining Black's mobility within the boring closely-knit defensive structures that the Russians tend to like. If this is so, then their similarity simply amounts to a shared bit of analysis. The work you do in conducting such analyses will be encoded in the dependency network. Every thought you've ever applied to a position tries to apply itself to each new position; perhaps that's holism enough.

> The logic-gates model of dependency networks makes us into rigidly logical thinkers.

This is a level confusion. We have seen that the reasoning encoded in a dependency network can be as illogical as you like. Another form of the objection claims that digital hardware must be rigidly logical in some deeper sense. It is hard to answer this objection because it isn't an argument, only an association of metaphors. Perhaps this confusion is excusable, given that the people who invented digital logic made the same association—they *wanted* to be rigidly logical but they didn't understand that it can't be done.

> The insight encoded in a given justification can only be reused in same situation it was recorded in.

How true this is depends on what it means to be the 'same situation'. The whole idea of dependencies is to define the loosest possible notion of sameness: two situations are 'the same' just in case they share the salient features, where saliency is determined by what you're trying to do. Whether this works out as intended depends on your representations. If every thought you ever think comes with a timestamp on it—e.g., "Mary loves John 01:32:16 11/21/86"—then you'll accumulate new justifications at an incredible rate and never use any of them. Fortunately, your brain probably can't tell time. A representation scheme can make dependencies useless in other ways as well; we'll return to these in Chapter C2.

> Your head will overflow if you record every new thought.

This, too, depends on issues of representation and dynamics. How many new thoughts do you really think? What counts as a new thought depends on what your representations are like. How likely is a new justification to be useful again? This depends on the ways in which everyday life is routine, a dynamic question. If everyday life is largely routine, then your head had better be big enough for you to be skilled at all of it.

> The permanence of dependencies falsely predicts that you'll never forget anything.

If forgetting 'something' meant no longer having 'it' in your head then this would be true. (Speaking of a thing as remembered or forgotten is a cheap way to add it to one's psychological ontology. One should resist this bargain.) Except in cases of organic brain damage, I don't think you actually do forget anything. Maybe you will never be reminded of something ever again, but this is a complex dynamic fact, not a fact about your mental inventory. One shouldn't think that 'remembering' and 'forgetting' are single, unified phenomena. Dependency maintenance offers a simple account of why you never forget how to ride a bike: the circuitry is always there, ready to go, even if your body has changed in small ways so that the calibrations have gotten rusty. The fact that we are often reminded of things we've long forgotten certainly doesn't prove that it's all still there, but why insist on the opposite?

> Why two values, IN and OUT? Why not three, or seventeen, or the whole real numbers?

Indeed, there exist more-or-less obvious generalizations of dependency maintenance to larger sets of values. One popular proposal is to distinguish YES, NO, and DON'T-KNOW. Another is to represent either 'certainty' or 'likelihood' (it is often not clear which) as a real number. I've posited two values out of simple parsimony. As soon as the developing dynamic theory compels me toward these generalizations I will certainly consider them. For me anyway, 'uncertainty' doesn't seem like a unified phenomenon. Even if it were, real uncertainty doesn't seem anything like the real numbers. Maybe I'm wrong.

> Surely we garbage-collect.

This idea confuses me. A Lisp interpreter can only collect garbage because its interactions with the outside world are so stereotyped that it can prove that certain structures will never influence output. A garbage collector has to start marking from somewhere, but how could it start marking from the outside world? If a thought was useful in the world once, how can you prove that a similar situation will never again arise?

Chapter B3

The initial implementation

B3a Context and summary

This chapter concerns a computer program called the *running argument system.* When I wrote the first simple version of it in 1983, all I knew for sure was that dependency maintenance is a way of accelerating the workings of a conventional rule system. When a rule runs, a justification records that the rule and its trigger are reasons for believing in its conclusion. The next time the same trigger matches the rule, the justification will rederive the conclusion automatically. This way, a simple gate propagation saves the effort of firing the rule again: instantiating the rule's right-hand side, building a new proposition, and installing it in the database. In the vocabulary of Chapter B2, this machinery is the 'source' and rule-firings are 'thoughts'.

The border between Parts B and C of this report corresponds roughly to my understanding in the middle of 1985, after two years spent trying to make the running argument system into a plausible cognitive architecture. Part B succeeds if it reproduces half the confusion I felt then. (My having told you about dynamic theory spares you the worst of it.) Four major revisions subdued the implementation issues, meshing the rule system and dependency system properly by assuring that all the right rules run with only a small constant overhead. This is reassuring as an existence proof, but since I don't think we have a rule system in our heads it's not a central point. This chapter describes these implementation issues.

The real problem comes in connecting a rule system to a simulated world (much less a real one) and trying to program it. Chapter B4 describes how this interaction ought to proceed. The dependency network is very much faster than the rule system, so one should write rules that needn't be continually firing on new triggers. My training in programming had definite opinions about how to write these rules. The standard methods work for a while; the implemented examples of Chapter B5 demonstrate all the virtues of dependencies that Chapter B2 has described. This is promising. In the end, though, one cannot reconcile these methods with the plain impossibility of using dependencies to support the standard account of abstraction, viz., variables and

constants. Chapter B5 poses this problem and Part C resolves it.

One might take two views of the running argument system. On one view there are rules being accelerated by dependencies. On the other view there is a dependency network being incrementally built from convenient but incidental parameterized specifications. You're on the road once the second view becomes natural. We haven't got a rule system in our heads. We can only properly ask where new dependencies come from once we understand how one's already-accumulated network leads one to interact with the world.

B3b Organization

The running argument system started as part of a (never-completed) animation system at Atari Research. (This was a project with Ann Marion and Jim Davis. I will grossly simplify the parts that aren't related to our main concerns here.) It provided a language, called Life, for writing rules about the critters and things in a cartoon world. A user could define the laws of the cartoon world, set up a cast of characters, arrange some initial conditions, and turn the system on; and life would go by. A drawing program could peek at the world at regular intervals to find out what to draw. Our initial goal was to animate Aesop's fable of The Boy Who Cried Wolf. This turns out to be hard. The hardest problem is that life doesn't naturally form stories. Left to its own, life falls into a routine. Days start and end, shepherds tend flocks, townspeople go about their business, wolves sneak in, shepherds sound alarms, wolves get chased away, townspeople head for pubs and sit around swearing at wolves, and so on. This is the stuff life is made of, day after day. To make anything *happen* against this background, storytellers have to intervene and say "however, on this particular day the shepherd had an idea." These were the hardest rules to write, the ones we never felt comfortable with.

This early version of the system had two parts, a rule system and a dependency system. These two systems shared a database of propositions (Lisp list structures), some of which had values of IN or OUT. The user would write rules in the Life language. (I'll describe the language and its semantics in a moment.) Whenever an IN proposition (the *trigger*) matches the *pattern* of an IN rule, the rule *fires* and the appropriate *consequence* is declared to be IN. If necessary, the system first builds the consequent proposition and inserts it in the database. Once the system was turned on, it continued running until there were no more rules to fire. In AI terms, this is roughly an ordinary forward-chaining rule system. (It is *not* a production system, as we'll see.)

The machinery underlying the Life language is designed on the assumption that most rule-firings have happened before. There are many applications in which this assumption would not apply. It seems plausible for the purpose of simulating everyday activity because life is routine. Most things that happen have happened before. Consequently, unlike advanced production systems like OPS5, the process of firing a life real has not been highly optimized. The dependency system does most of the work. Whenever a rule fires, the rule system makes a justification, declaring that the rule and the trigger

are reasons for believing the conclusion. That rule need not fire again on that trigger.

To get an idea of what the system does, let us anticipate the more detailed explanations by considering a simplified example. Here is a rule:

```
R13: (if (sees the-shepherd the-wolf)
         (rings the-shepherd warning-bells))
```

One sunny Monday, the first wolf appears (either as the consequence of a rule or as a premise):

```
A27: (sees the-shepherd the-wolf)
```

Then rule R13 fires on proposition A27. The rule has not fired on this trigger before, so the system builds a new proposition:

```
A28: (rings the-shepherd warning-bells)
```

and a new justification:

```
J41: (<= A28 (in R13 A27))
```

which we would draw as an AND gate connecting R13 and A27 to A28. Once the wolf goes away, A27 will go OUT. Assuming A28 has acquired no other justifications, A28 itself will go OUT as well. Now Tuesday comes and the wolf appears again, so that A27 comes IN again. Assuming rule R13 is still IN as well, A28 will also come IN, courtesy of justification J41. Whereas on Monday the system had to do some pattern matching and assemble a new proposition and a new justification, on Tuesday it only had to propagate a changed value through a gate. (Later we'll see how this works.) If the wolf makes daily visits to the shepherd's flock, A27 and A28 will bounce IN and OUT every day. On Monday it will take a moment to draw the correct conclusions, but on Tuesday it will happen right away. This effect is impressive in the large; the system accelerates as everything that happens often happens once.

The original system, then, had two modules (a rule mechanism and a dependency mechanism) sharing a collection of propositions arranged in two tightly intertwined datastructures (the database and the dependency network). Later, though, its purpose changed. In my own work I wanted to design a central system for an agent living in a world. Whereas originally the Life system simulated an entire world, now I considered using it as the central system for a single simulated agent. Though they differ in many ways, these two applications can both benefit from dependency maintenance because they are both routine. Just as most things that happen in the world have happened before, most things that happen in your head have happened before, or so I figured.

Since the Life machinery was only a central system, it needed two more components, a periphery (visual and motor apparatus) and a world simulation. Some of the propositions in the database serve as inputs from the periphery and others serve as outputs to the periphery. The running Life program now closes a loop with the outside world. The Life program together with its periphery and world simulation is called the *running argument system*, for reasons explained in Chapter B4. Here is its algorithm:

1. The world simulation updates itself.
2. The periphery computes new values for the central system's inputs, which represent perceptual information.
3. The periphery compares the new input values to the old ones. Any newly IN input is declared a premise; any newly OUT input is declared no longer a premise.
4. The central system now propagates dependency values and runs rules. Both the dependency system and rule system continue to run until they have both settled.
5. The periphery inspects the values of the central system's outputs, which represent motor commands.
6. The periphery and world simulation together arrive at (a) a set of proprioceptive propositions (judgements about the success or failure of the agent's primitive actions) and (b) a set of motor effects (the immediate physical consequences of the agent's actions).
7. The world simulation updates itself again, and so on ad infinitum.

This simulation does not bother constructing retinal images and muscle control signals. Instead, the periphery and world simulation work together to provide the effect of an agent getting about in a world. Although one should be suspicious of all simulated worlds, this cooperation in itself is not cheating because I am only studying the central system, not the rest. Some day, of course, it would be nice to connect the running argument system to the real visual-motor hardware of a real robot.

(It so happens that the periphery and part of the world simulation are implemented as another set of Life rules. Also, the modules and their connections are sufficiently abstract that they can be arranged in arbitrary configurations. I will suppress these extra generalities because they play no role in the examples. As a result, this chapter's description differs considerably from that in (Agre 1985). The program has evolved since then, but its outward behavior has not changed. Both this description and the one in (Agre 1985) are both correct as far as they go.)

B3c Example

I decided to test the running argument system on the blocks world. Like most of my predecessors, I chose the blocks world not because it is representative of ordinary human activity but because I could write the simulation and graphics code in about an hour. Yet the blocks world was about the worst domain I could have chosen. In fact, as we will see at the end of Chapter B5, the blocks world is so awful that there is no better way to expose the shortcomings of a lot of traditional methods than to try using them to get the running arguments system to work in the blocks world. Along the way we

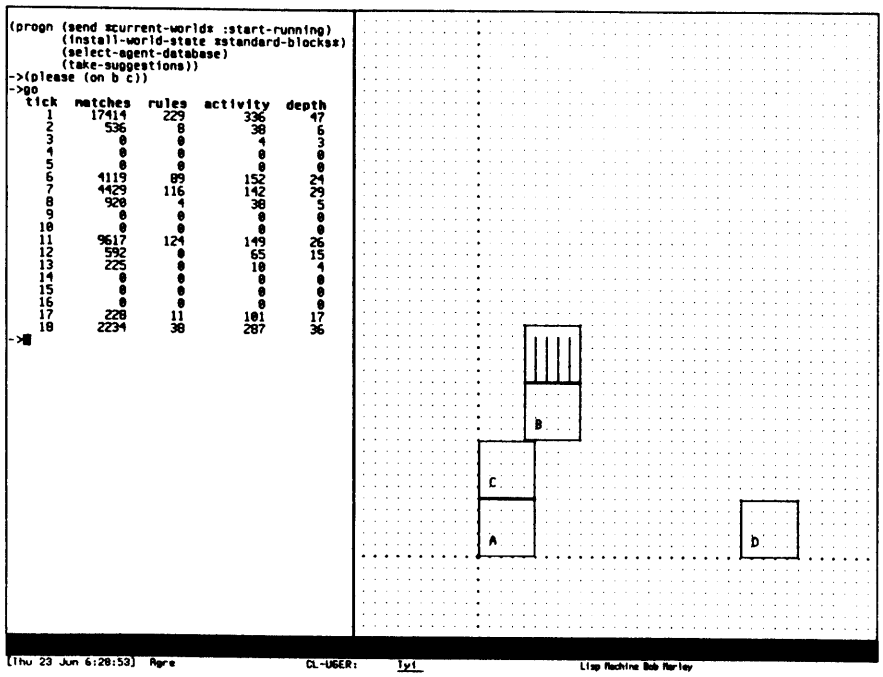

Figure B3.1. The system has been asked to put B on C. Since it has never been asked to do anything before, this task requires the system to run a large number of rules.

will encounter most of the basic structural and dynamic ideas we will need later on. In the end, I must explain how AI's blocks worlds obscure the nature of ordinary activity by falsifying the nature of playing with blocks.

Figure B3.1 shows the running argument system being asked to stack some blocks. In Chapter B5 we'll be seeing these screen snapshots again. For now let's just get the idea. The demonstration is extremely simple. The world, such as it is, consists of a table, a hand, and four blocks labeled A, B, C, and D. I'm asking the system to put block B on block C twice, starting from the same position each time. As we can see in Figure B3.2, it doesn't happen any differently the second time, only faster. Chapter B5 will go through these two demonstrations in much more detail.

Before we can discuss the dynamic aspects of the running arguments system in Chapter B5, we need to develop some vocabulary, in Chapter B4. The remainder of this chapter's examples will be highly simplified versions of the rules from the wolf story, designed to be as self-explanatory as possible. These rules illustrate only the machinery, not how anyone would consider programming it.

B3d Rule language semantics

The semantics of the Life rule language is neither procedural nor declarative. There is a database of propositions, each of which is a Lisp list structure composed of list cells,

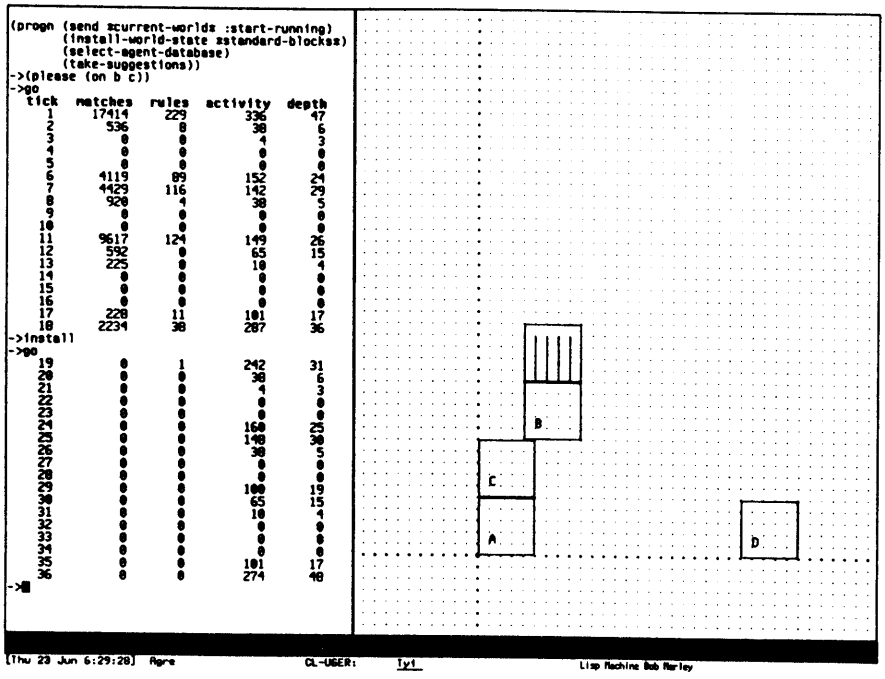

Figure B3.2. The second time it happens much faster. The dependency network is still very busy, but almost no rules need to be run.

symbols, and variables. Variables look like `?x`. Each proposition has a value, either IN or OUT. A proposition is OUT by default. Here are some possible (though hardly exemplary) propositions:

```
(forgets moose (flies squirrel))
(language is the house of being)
(for-all ?x (implies (human ?x) (mortal ?x)))
(((?x mortal) (?x human) implies) ?x for-all)
(plan put-on (?x ?y)
  (preconditions (clear-top ?x) (clear-top ?y))
  (actions (pick-up ?x) (move-to ?y) (put-down))
  (postconditions (cleartop ?x) (on ?x ?y)))
```

For technical reasons, two propositions are considered identical if they are the same except for the names of variables.

Some of the propositions are rules. There are two kinds of rules, IF rules and UNLESS rules. Rules are the only propositions with a prescribed syntax. Their syntax and semantics are as follows:

(if *pattern consequence-1 ... consequence-n*)

"So long as *pattern* is IN, make each *consequence-i* IN as well."

(unless *pattern consequence-1 ... consequence-n*)

"So long as *pattern* is OUT, make each *consequence-i* IN."

In each case, *pattern* and each *consequence* is a proposition. These propositions are likely to have variables in them. The critical phrase is "so long as." Let us consider some examples.

The example we considered earlier:

```
(if (sees the-shepherd the-wolf)
  (rings the-shepherd warning-bells))
```

should be read:

"So long as the shepherd sees the wolf, the shepherd rings warning bells."

As we saw, when the rule's pattern comes IN, the consequences come IN as well. (This rule, like most, has only one consequence.) When the pattern goes back OUT, each consequence goes OUT as well, assuming it is not being supported by another rule.

Rules can include variables. Variables are interpreted differently in the two kinds of rules. We might write:

```
(if (sees ?anyone the-wolf)
  (rings ?anyone warning-bells))
```

"Whoever sees the wolf rings warning bells."

In general, an IF rule fires when (a) the rule is IN and (b) some proposition matching the rule's pattern is IN.

Thus if we assert (meaning, establish as a premise):

```
(sees the-farmer the-wolf)
```

then the new proposition

```
(rings the-farmer warning-bells)
```

will come IN as well. If forty people see the wolf, the rule will fire forty times and all forty people will ring warning bells. If the wolf passes out of sight of ten of those people, those ten will stop ringing their bells even if the other thirty continue.

In general, an UNLESS rule fires when (a) the rule is IN and (b) no proposition matching the rule's pattern is IN.

We might write:

```
(unless (rings ?someone warning-bells)
  (sneaks-toward the-wolf the-sheep))
```

"So long as nobody rings warning bells, the wolf sneaks toward the sheep."

If this rule is asserted and nobody is ringing warning bells, then the rule's consequence will be IN. As soon as someone starts ringing warning bells, it will go OUT again. As soon as everyone stops ringing warning bells, it will come IN again.

Since rules are propositions, a rule's consequences might include other rules. We might write:

```
(if (owns-sheep ?person)
  (if (hears ?person warning-bells)
    (runs-to ?person the-meadow)))
```

"Anyone who owns sheep and hears the warning bells runs to the meadow."

```
(if (sneaks-toward the-wolf ?sheep)
  (if (is-a ?sheep sheep)
    (unless (shoots-at ?person the-wolf)
      (grabs the-wolf ?sheep))))
```

"If the wolf is sneaking toward a sheep then unless somebody shoots at the wolf it'll grab the sheep."

Abstract definitions and English paraphrases aside, the rule language is best thought of as a way of specifying dependency networks. An IF rule defines an AND gate each time it fires. If an IF rule's pattern has no variables, it will define a single gate. If it does have variables, it will define a separate AND gate for each matching proposition. See Figure B3.3. An UNLESS rule defines a single AND-NOT gate; if there are many propositions matching its pattern then each will be assigned a NOT input. See Figure B3.4.

It is best to learn to think in terms of both rules and networks. A complex rule defines a sort of template. Every binding of its variables will produce a new patch of dependency network. Sometimes we are interested in what the rules say and sometimes we are interested in the structure of the network. In particular, the semantics of the rules is quite independent of the use of dependencies to recapitulate rule-firings.

Life rules are not productions. The running argument system differs from production systems like OPS5 (Forgy 1981) in several ways. (Chapter C5 discusses production systems in more detail.)

1. All rules fire whenever they can. The architecture neither defines a notion of conflict nor provides mechanisms for conflict resolution.

2. A rule is not an imperative. It does not say 'when' but 'so long as'. It does not simply make its consequence IN, it also arranges (through the justification it creates) for the consequence to go OUT again when it is no longer supported.

3. There is no working memory as distinguished from the database as a whole. All IN rules can fire and all IN propositions can trigger rules. (This is true in practice for many users of production systems.)

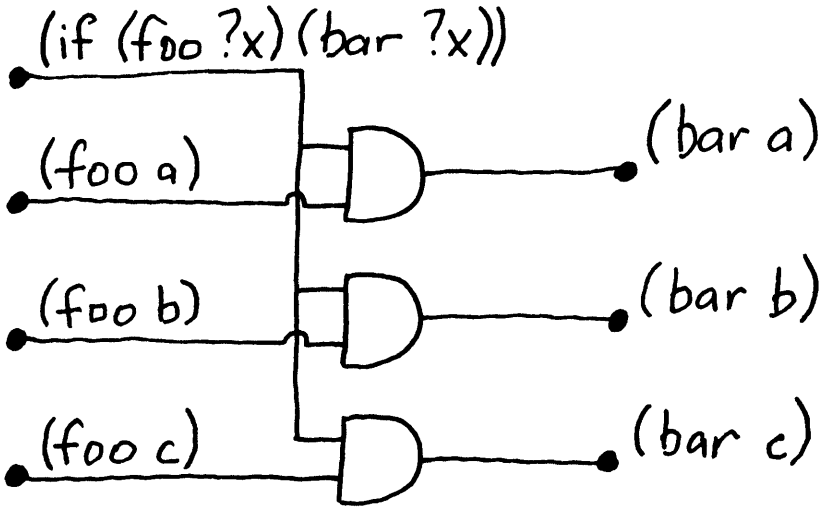

Figure B3.3. If an IF rule has a variable in its pattern, then many propositions might match it. Each one of the resulting rule firings will generate its own AND gate.

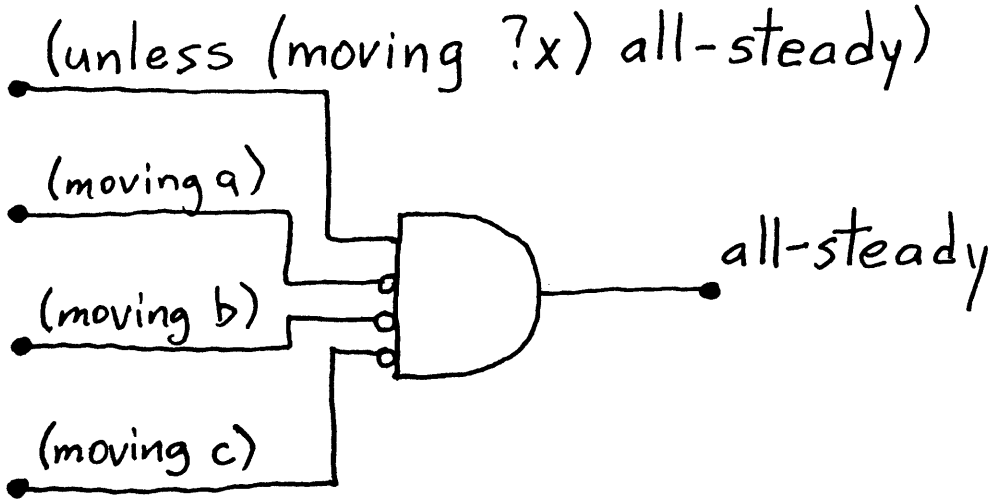

Figure B3.4. If an UNLESS rule has a variable in its pattern, then many propositions might match it. Each one of those propositions will get its own inverted input into the rule's AND-NOT gate.

4. The speed of the system does not depend on the speed of the rule-firing machinery. Most of the work is done in the dependency network, which is easy to implement in massively parallel machinery.

5. Life's rule-firing machinery is not part of any proposed cognitive architecture. Where a production-system theory might propose that someone has a certain set of productions in their head, a running-arguments theory would propose that they have a certain combinational logic circuit in their head.

The running argument system shares much of the motivation and spirit of production systems. Indeed, many people find the differences between them immaterial. I have many philosophical disagreements with this view, but my technical disagreement centers on the issue of variables and the distinction between imperative and so-long-as rule semantics. I will return to both issues.

Life programming is more similar to logic programming languages (Kowalski 1974, Warren 1983), particularly in the semantics of variables. Also, UNLESS rules differ from logical negation in the same way as negation in logic programming; each means 'not (yet) derived' instead of 'not derivable' or 'not true'. However, there are some differences:

1. The language defines no notion of predicates or functions. Propositions do not need to follow any syntactic rules unless they begin with **`if`** or **`unless`**.

2. There is no unification; in order for a proposition to trigger a rule it must match the rule's pattern, that is, there must be some assignment to the pattern's variables that produces the triggering proposition.

3. Whereas most logic programming languages are backward-chaining the Life machinery is forward-chaining. Also, whereas no logic program would ever derive all its logical consequences, Life continues firing rules so long as there are rules that can fire.

4. As with production systems, the efficiency of a Life rule set does not depend on the speed of pattern-matching. Once the system gets going, the dependency network does most of the work.

Again, I will return to the issue of variables.

B3e How it works

We have seen that the rule system is inside a loop. On every tick of the clock, the periphery updates the values of the perceptual inputs, waits for the rule system and dependency system to settle, and acts on the values of the motor outputs. On the first few ticks, the rule system is slow to settle because many rules are firing for the first time. Once the system gets going, though, rule firings become less common. Often

the perceptual inputs don't change at all. When the inputs do change, the changes usually propagate through the network without causing any new rules to fire. When this happens, we would like the rule system to settle very quickly, befitting the speed of combinational logic. Consequently, the rule system's interface to the dependency system must satisfy the following contract:

(1) A rule only fires once on a given trigger.
(2) It takes almost no time to determine whether any rules need to fire.

The first condition is easy enough. Every rule has a list of triggers it has fired on. Given a candidate rule and trigger, the rule system first checks this list and only proceeds if the trigger is not on it.

The second condition is harder. This section describes a simple mechanism, the outlines of which were suggested to me by David Chapman, that does fairly well. The next section (which is unpleasant and optional) describes a tricky refinement of my own that does quite well.

In all the implementations, the database is organized as a lattice of propositions under generalization. Here is what this means. Given two propositions $P1$ and $P2$, $P1$ generalizes $P2$ if there is some assignment to $P1$'s variables that produces $P2$. Here are some examples.

```
(rings ?x warning-bells) generalizes
(rings the-farmer warning-bells)

(eats ?x ?y) generalizes
(eats wolf sheep)

(eats ?x ?x) generalizes
(eats fish fish) but does not generalize
(eats cat fish)

(eats ?x ?y) generalizes
(eats ?x ?x)

(knows ?p ?x) generalizes
(knows ?p (sees ?q ?p)) which in turn generalizes
(knows the-wolf (sees ?q the-wolf)) which in turn generalizes
(knows the-wolf (sees the-shepherd the-wolf))
```

A proposition without variables generalizes nothing and the proposition `?x` generalizes everything. Generalization is a partial order because it has no cycles. But it is not a total order because it is common for a proposition $P1$ to generalize both $P2$ and $P3$, neither of which generalizes the other. For example:

```
P1: (knows ?p ?x)
P2: (knows the-wolf ?x)
P3: (knows ?p (sees ?q ?p))
```

Often the propositions $P2$ and $P3$ will both generalize some further proposition $P4$:

```
P4: (knows the-wolf (sees ?q the-wolf))
```

In such cases we speak of *reconvergence* in the lattice. Reconvergence raises hell with attempts at massively parallel implementation of a large class of symbolic indexing schemes, including the lattice technique.

To be technically accurate we should amend the definition to state that a proposition does not generalize itself. Figure B3.5 shows a sample lattice. The program's datastructures only record immediate generalization relationships, in technical terms the 'minimal generalizations' or the 'cover' of the relation. The algorithm that 'indexes' new propositions into the lattice is subtle and tolerably fast. (Unfortunately, it turns out that there is no simple way to index many propositions in parallel.) In the blocks world examples, a typical lattice has a few thousand elements.

The rule system's algorithm uses the proposition lattice. The lattice includes every proposition that has been made a premise or derived by firing a rule. In particular, it includes the rules themselves. It also includes a large number of 'propositions' that have no values. Among these are the patterns of all the rules. Thus if the rule

```
(if (sees ?anyone the-wolf)
  (rings ?anyone warning-bells))
```

is in the lattice, then so is the proposition

```
(sees ?anyone the-wolf)
```

even though it makes no sense by itself and is presumably neither IN nor OUT. As Figure B3.6 shows, the proposition lattice has a remarkable property. If P is the pattern of rule R and T is another proposition, T is a potential trigger for R just in case P is above T in the lattice. This suggests a simple algorithm for the rule system. For IF rules, the algorithm is:

- Whenever a proposition T comes IN, climb up the lattice from it. Whenever you encounter a proposition P that is an IF rule R's pattern, if R is IN then fire R on T (unless R has already fired on T).
- Whenever an IF rule R comes IN, climb down the lattice from its pattern P. Whenever you encounter a proposition T that is IN, fire R on T (unless R has already fired on T).

To fire an IF rule R on proposition T,

1. Match R's pattern to T, obtaining a list of variable assignments.
2. Perform these assignments on each of R's consequences, producing a list of new propositions $C_1 \ldots C_n$ (typically there is only one of them).

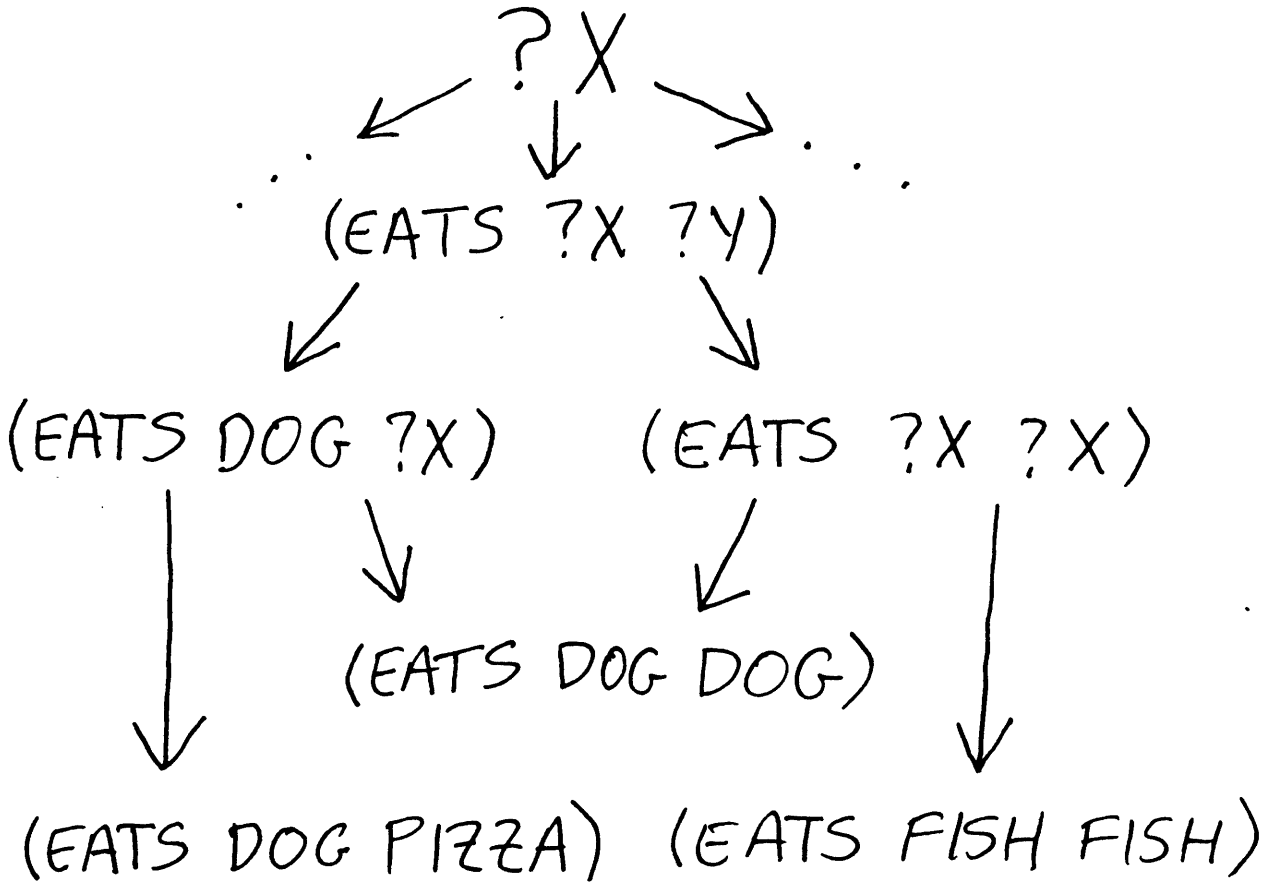

Figure B3.5. The patterns in the rule system's database are stored in a lattice.

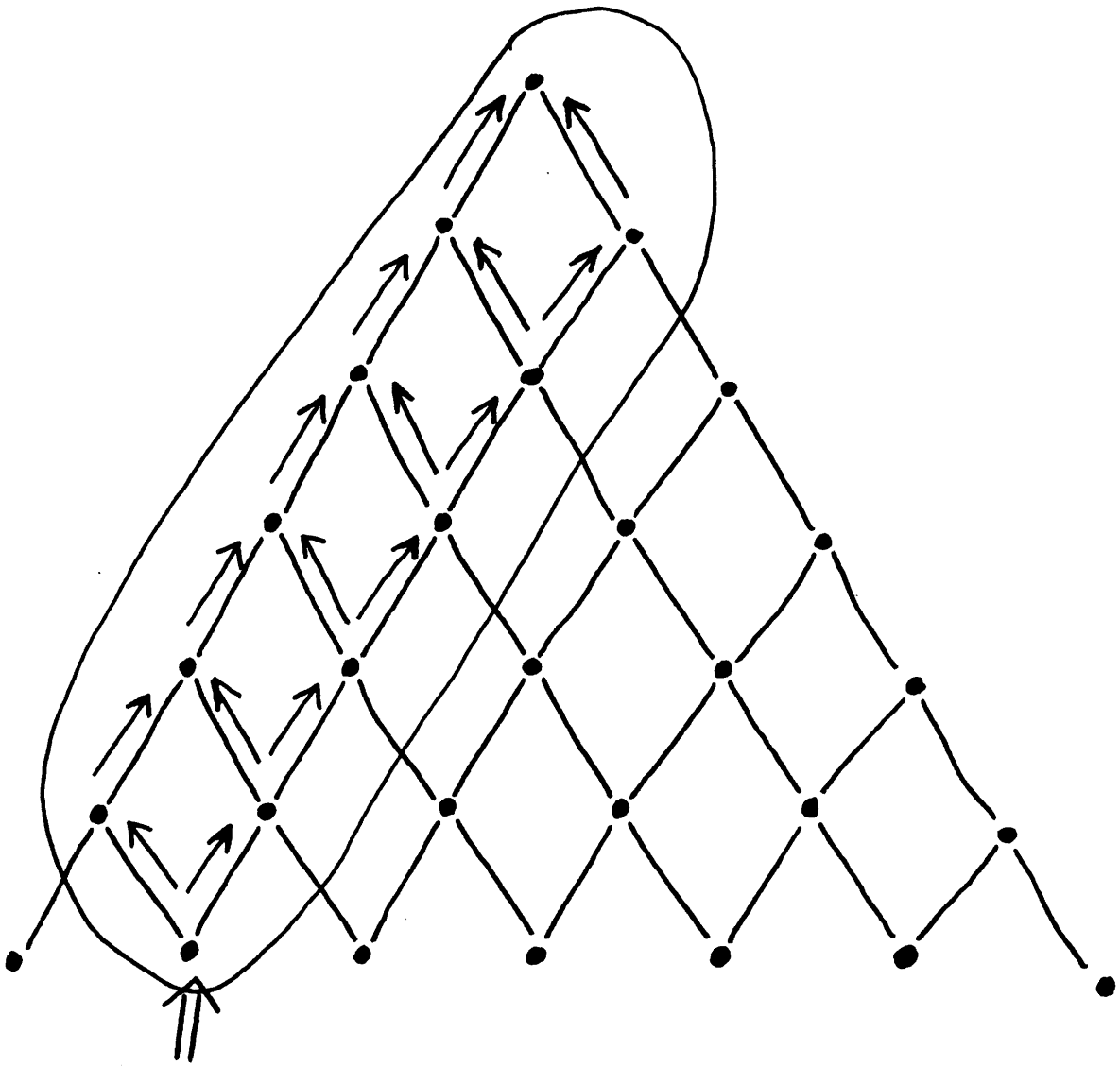

Figure B3.6. To find the patterns of all the rules that a given proposition might possibly fire, one need only move upward in the lattice starting from that proposition.

3. Index each of the new propositions into the lattice. (Some of them might already be there.)

4. For each C_i, construct a new justification: `(<= `C_i` (in R T))`.

The algorithm for UNLESS rules is not so intuitive and you will do well to skip it. (It took three tries to get it right.) An UNLESS rule 'fires' as soon as it first comes IN. From then on, whenever a new proposition matches the rule's pattern, it 'unfires', meaning that the new trigger is added to the OUT-list of the justification the rule created to support its consequence. The algorithm for an UNLESS rule of the form `(unless` P $C_1 \ldots C_n$`)` is:

- Whenever a proposition T comes IN, climb up the lattice from it. Whenever you encounter a proposition P that is an UNLESS rule R's pattern, unfire R on T (unless R has already unfired on T).

- Whenever an UNLESS rule R comes IN for the first time, climb down the lattice from its pattern P. Along the way, accumulate a list of every proposition Ti that has ever been IN. Finally, make a justification for the each of rule's consequences: `(<=` C_i `(in R) (out` $T_1 \ldots T_k$`))`.

To unfire an UNLESS rule R on proposition T,

1. For each C_i, find the justification that mentions R. (In practice, of course, they will all share a single AND-NOT gate.)

2. Add T to its OUT-list so that it reads: `(<=` C_i `(in R) (out` $T_1 \ldots T_k$ T`))`.

It is worth noting that no lattice-climbing occurs when a proposition goes OUT. If the OUT-going proposition enters into the support of any other propositions, the dependency system will take them OUT too if necessary.

If it seems expensive to climb around in the lattice, remember that every proposition one encounters while climbing down is a potential trigger and every proposition one encounters while climbing up is a potential rule pattern. Little effort is wasted. This version of the algorithm, unfortunately, requires climbing around the lattice every time a proposition goes from OUT to IN. The refinements in the next section repair this.

(A small technical note. Although the generalization relation does form a lattice once we go through technicalities like defining a unique minimal proposition, truth in formalization requires me to point out that the algorithm only requires generalization to be a partial order. Not all partial orders form lattices; in particular, the generalization relation probably doesn't form a lattice when restricted to the set of propositions that has actually been indexed at a given moment.)

B3f How it works (details)

This section is optional.

In reality, the system is more complicated. Recall that the goal is to permit the system to decide very rapidly whether any rules need to be fired. The system described in the previous section does not achieve this goal because the system must perform a search in the lattice every time a proposition goes IN or OUT. I will describe the algorithm only for IF rules because the methods for UNLESS rules are too complicated to state but still easy enough to rederive.

At a small expense of space, the system can avoid doing any search except the first time a given proposition comes IN. Every proposition keeps a bit indicating whether it has ever been in. When a proposition comes IN for the first time, it searches upward in the lattice; when a rule comes IN for the first time, its left hand side searches downward in the lattice. While searching, it looks for rules that might be able to fire now. But it also looks for *potential rule firings.* When a trigger is searching upward for patterns, it looks for rules which are not currently IN but have been IN at some point in the past. Likewise, when a rule's left hand side is searching downward for triggers, it looks for propositions which are not currently IN but have been IN at some point in the past. All of this information is stored with the proposition doing the searching.

Thus every proposition has a list of potential rules and every rule has a list of potential triggers. Whenever a proposition comes IN or goes OUT , it checks all of its potential rules to see if they are IN. If any are then the rules fire and are removed from the proposition's list of potential rules. Whenever a rule comes IN or goes OUT , it checks all of its potential triggers to see if they are OUT. If any are then the rule fires on them and they are removed from the rule's list of potential triggers.

This algorithm works well in practice because the lists of potential rules and triggers are always short. One could write pathological rule sets in which the lists were choked with rules and triggers that could lead to useful work in principle but never will in practice, but this has never happened with any actual rule set.

One additional detail concerns the pattern lattice itself. During a large run of the system, the lattice typically contains five thousand propositions. All of the algorithms are carefully designed to search only the minimally necessary regions of the lattice. Still, some steps of those algorithms take time proportional to the fan-out of the propositions they visit. If some proposition has a hundred other propositions immediately below it in the lattice, the algorithm has to make a hundred separate decisions about which branches to pursue. Almost all of these decisions are typically negative. Consequently, it is best if the fan-out in the lattice is kept to a minimum.

The problem is worst near the very top of the lattice. The top element of the lattice is a plain variable, `?x`. Unless care is taken, its immediate offspring will be an extremely long list of left hand sides of various rules. The problem can be alleviated by indexing such intermediate patterns as `(?x ?y)` and `(if ?x ?y)` into the lattice as well. Though these patterns have no meaning to the system, they partition the offspring of `?x` without

changing the behavior of the algorithm.

Unfortunately, the problem of excessive fan-out also occurs at many other places in the lattice. For example, any pattern with two variables, such as `(on ?x ?y)`, is liable to accumulate as offspring all possible instantiations of itself, such as `(on c a)`. The problem is gets rapidly worse as the rules become more complex; many of the patterns in the database involve upwards of a hundred list cells. This effect can slow the system by a factor of ten. This slowness has no theoretical import (because I don't believe that people have rule systems in their heads) but it does make it impractical to debug substantial rule sets.

I've developed a series of heuristic tools for dealing with the problem. It is an interesting technical puzzle to write an algorithm that, given a proposition and its excessively numerous immediate specializations, chooses appropriate intermediate specializations. These intermediate specializations should sort the offspring propositions into neatly balanced partitions. The intermediate propositions should be sufficiently natural that the partitions are likely to stay balanced as future propositions arrive. This puzzle turns out to be insanely difficult to solve acceptably. The current algorithm, most recent of a long series, covers several pages of code and does a poor to medium job.

This concludes the description of machinery underlying the Life language.

B3g Incremental updating

We can now leave the microscopic details of the running arguments system's machinery and begin considering dynamic issues, that is, what it does on a large scale in practice. The substantial discussion of dynamic issues begins in Chapter B4. We can best prepare for it by considering what Life programming is like independent of any theoretical scruples.

Many of the dynamic ideas turn on the system's ability to update itself incrementally when its premises change. When the system is interacting with a simulated world, the premises that change are the perceptual inputs. Rather than dive straight into these issues, let's take a simpler case of the same effect.

When you're debugging a rule set, the premises that matter are the rules. If you rewrite a rule, take the old version OUT and make the new version IN. The best way to debug is to watch the system running. When you don't like what it's doing, poke around in the dependencies.

Suppose we're debugging the rules for the wolf story. Here are some plausible cartoon premises:

```
R12: (if (sees ?anyone the-wolf)
         (rings ?anyone warning-bells))
R37: (if (in-town ?person)
         (if (rings ?anyone warning-bells)
           (hears ?person warning-bells)))
A46: (owns-sheep Katya)
```

```
R47: (if (owns-sheep ?person)
         (if (hears ?person warning-bells)
           (runs-to ?person the-meadow)))
```

(The labels— R12, A46, etc.—are assigned by the system to provide short names for the propositions. This example is fictional because the real rules would require too much explanation. We'll see plenty of real rules in Chapter B5. Also, keep in mind that our eventual aim is not to write cartoon simulations but to design central systems for agents.)

Now you're watching the system run. The story reaches its climax. The wolf appears and sneaks up on the sheep. The shepherd sees the wolf and rings the warning bells. The townspeople hear the bells and go running. That isn't right. Stop the system. Among the IN propositions is:

```
A63: (runs-to Katya the-meadow)
```

We can inspect the dependencies behind A63:

```
(why? A63)
A63 is in because R51 ran on trigger A39:
   R51: (if (hears Katya warning-bells)
            (runs-to Katya the-meadow))
     R51 is in because R47 ran on trigger A46:
        R47: (if (owns-sheep ?person)
                 (if (hears ?person warning-bells)
                   (runs-to ?person the-meadow)))
          R47 is a premise.
        A46: (owns-sheep Katya)
          A46 is a premise.
   A39: (hears Katya warning-bells)
     A39 is in because R49 ran on trigger A28:
        R49: (if (rings ?anyone warning-bells)
                 (hears Katya warning-bells))
          R49 is in because R37 ran on A40
             R37: (if (in-town ?person)
                      (if (rings ?anyone warning-bells)
                        (hears ?person warning-bells)))
               R37 is a premise.
             A40: (in-town Katya)
               A40 is a premise.
        A28: (rings the-shepherd warning-bells)
          A28 is in because R12 ran on A27:
             R12: (if (sees ?anyone the-wolf)
                      (rings ?anyone warning-bells))
               R12 is a premise.
             A27: (sees the-shepherd the-wolf)
               A27 is in because ...
```

And so on through the reasons behind the wolf being in the meadow and the shepherd seeing it. Even if there are a hundred rules among the premises (not at all unusual) this display presents only the ones that entered into the troublesome conclusion. Reading them, we find that the townspeople are too gullible. We've written rules that lead them to conclude that the shepherd is a liar, but we haven't made them act on that conclusion. Let's rewrite R47 to make it more general:

```
R68: (if (owns-sheep ?person)
       (if (believes ?person (at the-wolf the-meadow))
         (runs-to ?person the-meadow)))
R69: (if (rings ?anyone warning-bells)
       (if (hears ?person warning-bells)
         (unless (believes ?person (liar ?anyone))
           (believes ?person (at the-wolf the-meadow)))))
```

(What a kludge! It's an instructive exercise to try writing these rules as generally as you possibly can. When I gave up trying to write rules to completely capture the 'real reasons' behind the events in the wolf story, I was long past a hundred rules and diverging rapidly.)

When we retract R47, both R51 and A63 go OUT but everything else stays unchanged. When we now assert R68 and R69, they both run, deriving:

```
R70: (if (believes Katya (at the-wolf the-meadow))
       (runs-to Katya the-meadow)))
R71: (if (hears ?person warning-bells)
       (unless (believes ?person (liar the-shepherd))
         (believes ?person (at the-wolf the-meadow)))))
R72: (unless (believes Katya (liar the-shepherd))
       (believes Katya (at the-wolf the-meadow)))))
```

The system has already derived:

```
A54: (believes Katya (liar the-shepherd))
```

so rule R72 does *not* license the conclusion that Katya believes the wolf to be in the meadow. Finally we let the simulation proceed. Katya does not go to the meadow (nor does anybody else), the wolf eats the sheep, and the shepherd feels bad.

Normally when you fix a bug in a program you run the program all over from scratch, wasting seconds or minutes getting back to where the problem was. With dependency maintenance, the program is re-run incrementally. The system only does the work that needs to be done differently.

B3h Advanced rule-writing

This section contains technical details that will help explain some of the code fragments in the Chapter B4. It can be safely skipped.

Life is a simple language, but still perfectly general. It's Turing universal, for what it's worth, in two senses. One can write a Lisp interpreter in Life rules alone. It's not hard but it runs an enormous number of rules. The dependencies do provide automatic memo-izing, for what it's worth. On the other hand, if a Life system is connected to a world that behaves like the tape of a Turing machine, one can write Life rules for a universal Turing machine. These rules require no variables so there's a definite upper limit on the size of the resulting dependency network. The periphery or the world probably has to supply a two-phase clock signal to synchronize the discrete tape operations.

Unlike many rule languages, Life does not provide for boolean combinations of rule patterns. For example, one might like to rewrite rule R37:

```
old: (if (in-town ?person)
         (if (rings ?anyone warning-bells)
           (hears ?person warning-bells)))
new: (if (and (in-town ?person)
               (rings ?anyone warning-bells))
         (hears ?person warning-bells))
```

To allow such rules, one can take advantage of the fact that rules are propositions too:

```
(if (if (and ?p ?q) ?c)
  (if ?p (if ?q ?c)))
(if (if (or ?p ?q) ?c)
  (if ?p ?c)
  (if ?q ?c))
(if (if (not ?p) ?c)
  (unless ?p ?c))
```

(The actual rules are more complicated for reasons of generality and efficiency.)

One could even write rules that support backward-chaining.

```
(if (if-shown ?p . ?q)
  (try-to-show ?p)
  (if ?p . ?q))
(if (unless-shown ?p . ?q)
  (try-to-show ?p)
  (unless ?p . ?q))
(if (can-show ?p)
  (if (try-to-show ?p)
    ?p))
```

(See (Agre 1985) for a fuller discussion of these rules, which differ from the Amord versions (deKleer *et al* 1978) only in their simpler syntax.) For example, it is often necessary to constrain two variables to have different bindings. To express the idea of two different people seeing the wolf, one might say:

```
(if (and (sees ?a the-wolf) (sees ?b the-wolf))
  (if-shown (neq ?a ?b)
    ... ))
```

We can now write rules for showing equalities and inequalities:

```
(can-show (eq ?x ?x))
(if (try-to-show (neq ?x ?y))
  (try-to-show (eq ?x ?y))
  (unless (eq ?x ?y)
    (neq ?x ?y)))
```

Given these rules, if both Katya and Anna see the wolf, then the following series of propositions would get asserted, courtesy of these rules:

```
(if-shown (neq Katya Anna)
  ... )
(try-to-show (neq Katya Anna))
(if (neq Katya Anna)
  ... )
(try-to-show (eq Katya Anna))
(unless (eq Katya Anna)
  (neq Katya Anna))
(neq Katya Anna)
...
```

In practice the running argument system does not use backward-chaining in more complex ways than this.

Rules like these raise a serious question of what's fair in Life programming. If my psychological theory allows me to write arbitrarily sophisticated Life programs, it has little content. This question has no easy, complete answer, but a great deal follows from first principles. One should not write rules that must fire regularly (that is, with different variable bindings) in situations that ought to be routine. It is also a good policy not to write rules that fire on other rules except as a notational convenience. The issue will keep coming up in later chapters. Above all, keep in mind that our topics are routine activity and the interactions between an already-existing dependency network and its periphery and world. Life rule sets are not psychological theories but specifications for networks.

Chapter B4

Running arguments

B4a Context and summary

Chapter B2 has introduced a partial account of central system architecture and Chapter B3 has described an implementation of it. It might strike you as odd that there is so little to it, and so little that is original by any familiar standard. The way we ordinarily think about computer programming, the how-it-works, and usually the what-it-does too, can be read straight off the 'code'. Once you have described the machinery you're almost done. That is not true here. We're hardly off the ground. Most of the story is in issues of personality and dynamics, about which I have only provided hints. This chapter introduces some basic ideas about these things, starting with the dynamic idea of a running argument.

Running arguments are one way to think about what goes on in a dependency network when its owner is engaged in routine activity. According to this view, on every moment you conduct an argument about what to do now. (I am using the word 'argument' loosely and metaphorically.) The collection of proposals and objections and evidence and reasons pro and con that enter into the argument is known as the *argument structure.* All these components (each generically called an argument) have dependencies attached to them. The dependency network acts as a warehouse of all the arguments that have entered into all of one's past decisions. Each argument joins the argument structure when it is applicable and leaves when it is no longer applicable. Thus the argument structure is continually incrementally updated to reflect one's current goals and understanding of the world. This dynamic effect is called a *running argument.*

The slogan of running arguments (adapted from a saying of John Batali, who derives quite different conclusions from it) is:

> One must continually redecide what to do.

We might describe a running argument thus:

> A running argument is a continually incrementally updated hierarchical argument about what to do now.

There are several words to explain: argument, continually, incrementally, updated, hierarchical, and now. Above all, a running argument is a dynamic phenomenon; none of these words names an algorithm or a datastructure. Some simple ideas about the dynamics of running arguments suggest new views of what AI has called hierarchical, nonlinear, and incremental planning. The next three sections motivate the idea or running arguments by contrasting it with conventional AI ideas about the organization of activity.

Section B4b concerns the conventional notion of Planning. It describes the inflexibility of Plan-following and argues for a continual updating of the reasoning behind one's actions. Narratives from everyday life help dispel any intuitiveness of Planning in favor of running arguments.

Section B4c concerns the notion of hierarchy. AI Planning work has typically involved a recursive decomposition of goals into subgoals; this section outlines some of the changes this view must undergo as our prototypes of activity shift from planning-ahead to improvisation. Working out the details of these suggestions, however, is a topic for future work; this report will continue to assume the version described in Section B4c.

Section B4d concerns the notion of deliberation as it appears in Doyle (1980). Doyle describes a scheme of deciding-what-to-do-next by recursive argument and counterargument. I will argue that Doyle's scheme is overly centralized. Moving the burden of proof from the proposer of an action to an objector simplifies the architecture and allows for massive parallelism.

Section B4e puts the pieces together. It prepares for the demonstrations of Chapter B5 by describing several ways of thinking about the dynamics of a running argument.

B4b Planning vs knowing what you're doing

In discussing planning, we must not get caught up in a sterile opposition between 'planning' and phrases like 'just trying things' or 'muddling through' or 'figuring it all out as you go along'. This too-common practice contrasts a uselessly inflated notion of 'planning' with correspondingly anemic alternatives. We must not use the word 'planning' indiscriminately to refer to absolutely any deliberateness or intentionality or 'knowing ahead of time what you are going to do'. Instead, let us recognize two useful senses of the word. Lower-case 'planning' retains its ordinary vernacular use, that is, comporting oneself toward a (written or subvocalized) plan. AI research, by contrast, has been concerned with upper-case 'Planning', which distinguishes a smart Planning phase and a dumb execution phase. The Planning phase receives representations of a situation and a goal; in exchange it delivers a Plan. The execution phase carries out the Plan, to the letter, in a fashion one would call mechanical. One thinks of the Planning and execution phases as alternating—Plan, execute, Plan, execute, Plan, execute—an idea called *incremental*, or *interleaved*, Planning. (See Chien and Weissman 1975, Giralt *et al* 1984, McDermott 1978, Tate 1984, Wilkins 1988.) Chapter C5 discusses some other variations on the theme.

During its execution phase, an agent's Plan is directly responsible for its activity. The agent itself is almost asleep. It just cranks out the actions indicated by the Plan. This sort of blind action is obviously dangerous. Adding extra features to the executive can alleviate some of the more obvious hazards. For example, if one of the actions detectably fails or is detectably inapplicable, the executive can pass control back to the Planner—an idea called 'execution monitoring' (Fikes 1971, Ghallab 1985, Munson 1971). One should cease executing a Plan any time the next action it prescribes is irrational in context. Thus an executive should be approximately as smart as its Planner.

To use a plan, you also have to use some sense. The difference between a Plan and a plan is that Plans are made for dumb executives whereas plans are made for ordinarily intelligent agents. Perhaps we could get very far without plans, but they aren't what directly generate our activity. An executive can't do anything without a Plan; running arguments are my account of what we do *whether or not* we happen to be following a plan. An executive interposes only a simple automaton between its Plan and its muscles; running arguments are my account of what people interpose between *their* plans and *their* muscles. Dwelling for a few pages on the weaknesses of Planning may seem redundant, but it will deepen our understanding of moment-to-moment activity and stop us being seduced by next week's latest extension. At each step we will see the value of keeping dependencies on all the reasoning leading to one's actions. The section closes by describing running arguments as a limiting case of incremental Planning, where the increments have become vanishing small and the Planning-execution distinction has disappeared. Subsequent sections will elaborate on this description.

In analyzing Planning, the key words are *anticipation* and *trouble*. When a plan goes wrong, there must be have been something you didn't anticipate. One says that something 'came up'. A *contingency* is when you fail to anticipate something bad. If you fail to notice too many contingencies then you will get hurt. An *opportunity* is when you fail to anticipate something good. If you fail to notice too many opportunities then you aren't paying attention. (For an interesting discussion of opportunities see Birnbaum 1986.) There are things you can't anticipate and things it would be foolish bothering to anticipate; these call for *improvisation*. Let's consider an example.

> I was spending the afternoon making a demo videotape in the vision lab at Oxford. It had to be done by dinnertime so I could take it on a trip. Everything was going wrong. I had just returned from a trip home to pack. As I walked in the door, someone told me that S was looking for me, something about my trip. I wanted to set up the next program to be taped, but all the equipment was being used by people trying to fix things. So I decided to go talk to S, who was two flights up. Walking up I also decided I ought to stop by the bathroom, which is across from S's office. Approaching S's office I had to decide which room to head for. It occurred to me that I shouldn't risk missing S, so I decided on S's office. Arriving at S's office, though, I saw that she was on the phone. Instead of waiting I headed for the bathroom. For what it's worth, I believe I told myself something to the effect that I

> was less likely to miss S so long as she stays on the phone. But this wasn't anything I was aware of putting much effort into.

I am certainly not going to argue that 'Planning can't handle this case'. One could build an executive for which having to wait was sufficient reason to cancel the Plan and return control to the Planner (assuming the executive can figure out that you can't expect to talk to someone who is on the phone (unless it's an emergency (unless the phone call concerns a bigger emergency (etc)))). That is not the point. For every story I can tell about people changing course, one could make another entry on the executive's list of reasons to cancel Plans. Indeed, it is hard to think of a condition that could never be sufficient reason to abandon a Plan. In order to stay ahead of all these conditions, the increments between Planning steps will have to be pretty small.

So far so good. We could interleave Planning and execution at 10 Hz or so. But that would be a superficial response; there is more at stake. Even a brief look at the fine detail of everyday activity reveals a long list of reasons for an executive to appeal to the more complete intelligence of its Planner. The word 'Planning' begins to dissolve into an obscure way of referring to 'thinking' in general. The notion of Planning seems to run together two different, equally important, matters: 'planning ahead'—that is, using ideas about the future in deciding how to act—and figuring out what makes sense to do *now*. Much turns on your intuitions about the relative proportions of these two: how much you try to anticipate and how much you figure out as you go along. The word 'Planning' strongly prejudges this question in favor of exhaustive anticipation and negligible improvisation.

AI Planning research has generally considered domains in which everything can be anticipated. Since the Planner can't count on the executive to use any sense in carrying out the Plan, the Plan has to be guaranteed. If you don't know what might happen you can't Plan. Precise prediction is unnecessary so long as the Planner can characterize a space of contingencies, whether by enumerating them, forming a common abstraction from them, or describing a geometrical envelope around them (Lozano-Pérez, Mason, and Taylor 1984). Nonetheless, as the space of contingencies expands, it becomes less likely that any one Plan will be sufficient. Faced with such uncertainty, the Planner is under great pressure to play it safe. I once heard someone suggest formalizing Planning in terms of game theory: life is a game and the world is playing against you. This skid-row attitude sums up Planning quite well.

Life can be hard, but it's not hard in the way Planning is hard. We organize our activity without explicitly anticipating everything that might happen. The simplest conversation is an amazing dance of moment-to-moment give-and-take with a branching factor beyond calculation. A Planner confronting the world of everyday life sees a bewildering maze of possibilities. But everyday life doesn't *seem* like a maze. It seems perfectly tractable. All we can anticipate is that we'll generally be able to figure things out when the time comes.

What makes the world hard to anticipate? When you have to work in novel surroundings, you usually don't know where things are without looking. There is also the

frame problem, which makes it computationally difficult to anticipate the effects of your actions. Your knowledge of a thousand simple matters like the location of your coffee cup will grow rapidly out of date as you go about your tasks. Even when there is enough information, keeping track can be a waste of effort. You don't bother (I hope) keeping track of the arrangement of the gadgets in your kitchen's gadget-drawer. There is no use keeping track when you can simply look and see. The world, as we've observed, is its own best simulation.

Some examples might help. Here are twenty things it would rarely be worth trying to anticipate:

- Whether the handle on a refrigerator door is on the left or right.
- Where the chalk is located on the blackboard's chalk tray.
- How the aspirin tablets are arranged in the aspirin bottle.
- Whether any mail has come today.
- How your key ring will be oriented as you retrieve it from your pocket.
- Which section of the newspaper the comics will be in today.
- Where there will be a free seat in the subway.
- How all the dirty dishes will be arranged when it is time to wash up.
- How many bottles accumulated for the recycling bin last month.
- Which side of a record will be up when you remove it from its sleeve.
- Whether the record will need cleaning before you play it.
- How the spatula will be tangled among the gadgets in the gadget drawer.
- In what order the shirts are hung in your closet.
- Whether you will have to open another box of cereal.
- How many pennies you have in your pocket.
- Which way the handle will point when you are handed your pint of beer.
- How the other pedestrians will be distributed along the sidewalk.
- Which slots in a half-full egg carton have the eggs in them.
- When your watch battery will start running down.
- Where the knob on the oven is set from the last time it was used.

Some observations on this list, including some dynamic phenomena deserving further description and explanation:

- These things might be worth anticipating in another culture. Maybe a penny is a lot of money, aspirin arrangement is an omen, shirts are worn in a certain order, poison is kept in left-handed fridges, etc.
- For every entry on this list, one can imagine circumstances under which it would be worth trying to anticipating it. There are probably interesting patterns among the halfways plausible ones.

- In particular, when's and how-many's are often worth anticipating if they are going to be grossly outside their usual range. We have a good sense of normal ranges even though they are often hard to define.
- Some of these can be anticipated by taking the effort to 'keep track'. Keeping track of things is usually remarkably difficult. I'd like to know under what conditions it is relatively easy.
- Things that aren't worth anticipating are often hard to remember afterward. Most of them matter so briefly and are accommodated so easily that they don't seem to make much of an impression.
- Furthermore, one is rarely aware of explicitly declining to anticipate something. Maybe we do tons of unconscious declining. More likely we only try to anticipate what we think we have to.
- Equipment that nobody else uses will be easier to anticipate. People who live alone often know exactly how the cereal boxes in their cupboards are arranged.
- Sometimes you will find yourself anticipating things like these after using the same item of equipment in the same way under the same conditions very many times.
- Little is lost in trying to anticipate these things. Many regular subway riders have superstitious beliefs about where free seats will be found.

Above all, it matters whether everything that happens is caused by the agent. AI Planners have typically been demonstrated in inert domains. Who knows what other agents or processes might do? One might think about dealing with others in the way that games like chess are customarily formalized, using the familiar technical apparatus of alternating 'moves' from a constrained 'legal' set. Even when this sort of formalization is applicable, it often misses the point. Think of a game like go. Some kinds of philosophical disputation are made of stereotyped moves, but most of life isn't. You share sidewalks, roads, shops, kitchens, laboratories, and bars with other people. Even though most of those people are moderately responsible, all of them are only moderately predictable. If you live with other people, for example, they will mislay the can opener behind the breadbox, drink your last beer, leave all the forks sitting unwashed in the sink, occupy the shower when you'd like to be using it, tear Calvin and Hobbes out of the comics page before you've read it, get half a dozen phone calls at ten minute intervals, and otherwise foul up your plans. Such contingencies may be annoying, but rarely are they memorably disastrous.

Why ever plan ahead? Why not figure everything out as you go along? In planning ahead, you seek to avoid trouble. One idiom warns of 'painting yourself into a corner', a fate that another idiom blames on 'burning bridges'. There is an important trade-off: if you plan ahead instead of improvising you might avoid some trouble, but you will have a lot less information about the circumstances you will be acting in. The role we assign to planning in the dynamics of everyday activity depends on how impressed we are by everyday trouble. A Planner that insists on producing guaranteed-correct Plans is so impressed by trouble that it won't risk any trouble at all. It is easy to understand

why programmers should write such timid programs; how else could one's research be principled? By what *a priori* criterion could one's program determine certain risks worth taking and others worth avoiding? Without such a criterion, a heuristic Planner is in serious danger.

To understand the role of trouble in everyday life, we might start by asking how much trouble people actually get themselves into. Certainly everyone can cite horrendous episodes of trouble. People regularly burn themselves, lock themselves out of their houses, lose their plane tickets, get sick, get lost, get drafted, and run out of garlic. These are all notorious problems. They make good stories. But trouble is much more prominent as a conversation topic than as an actual feature of life. Trouble is simply more memorable than non-trouble. You might make breakfast a thousand mornings in a row—and remember only the one you dump in your lap. This bias suggests, correctly I believe, that trouble is less common than we make it out to be. But we still don't know *why* trouble is unusual. To understand the nature of trouble, we need a better understanding of the dynamics of everyday activity. For present purposes, we only need to consider the matter briefly. The next section will return to it.

Observation of everyday activity immediately reveals a great deal of the following:

> I was in a kitchen whose sink is immediately to the right of the oven. I had just put my lunch in the oven to cook and was standing at the sink washing dishes. While washing, I noticed something moving out of the left corner of my eye. It was smallish, white, wispy in appearance, and seemingly floating in the air over the oven, perhaps a little in front of it. Thinking it might be smoke from the oven, I turned to look at it. Focusing on it, I saw that it was a small white feather, perhaps a bit of down. No phrase I can come up with captures the experience very well. I had no sense of drawing conclusions or formulating doubts about this possible smoke, nor any sense of *needing* to do these things, in the moment before I turned to look. I just turned and looked. This pattern of glancing to inspect something in one's peripheral vision is exceedingly common. Sometimes one has no idea what the peripherally glimpsed thing is, but sometimes a complex set of guesses and arguments will occur to one in the same momentary flash in which I 'thought it might be smoke'. Presumably these guesses and arguments accumulate through experience, but I don't know if the dynamics of accumulations of peripheral visual interpretations differs from the dynamics of other accumulations.

> Often in writing I will begin writing the wrong word, or write the intended word incorrectly. Noticing this, I will scratch it out, write it correctly, and carry on. Similarly, typing on a computer I watch the characters trailing behind the cursor and very frequently (several times the average line) use the 'delete' or 'rubout' key to erase mistaken keystrokes. When more complex erasing commands are available (as in the text editor) I will often have occasion to rub out one or more whole words.

> I was in bed sick all day, getting up periodically to visit the bathroom. Night had recently fallen and though my room was lit the rest of the house was now dark. I got up to head again for the bathroom. Reaching the door of my room I came upon the fact that it was dark beyond. Hoping to make some light, I went for the switch—which of course was the light switch in my own room. Finding myself now in complete darkness I (in some order) realized what I'd done and reversed the switch. Walking into the hallway I now looked for and hit the hallway light switch. On subsequent occasions I was more prepared for the darkness in the hallway and navigated to the bathroom easily enough in the dim light. Whereupon of course I turned on the bathroom light, which was controlled by a cord just inside the door. Although my mistake was interesting in itself, the present point is simply that nothing bad came of it. This story perhaps overdoes the point, given that hitting a light switch is the epitome of a reversible action.

In each of these cases, something simple has happened. Trouble has arisen, been noticed, and been corrected, all without any break in the action. Sometimes the action breaks momentarily:

> This bathroom light cord provides an example of its own. It is about four feet long and hangs from the ceiling. Like many hanging control cords (especially on venetian blinds), at the bottom it has a smallish white plastic cone. (I suppose you are supposed to hold this cone when you pull the cord.) Unfortunately, the cord is made of fairly light string (a fact I never formulated till it first occurred to me to use it as an example). Thus when I reach for the string, I usually inadvertently strike it with my hand and send it swinging out my grasp. Once set it motion as a pendulum it has an amplitude of almost a foot and a period of several seconds, so I have almost no chance of finding it without looking directly at it, especially since it usually moves close along the wall. If I am walking in then looking at it takes a little work since it is dark (which of course is why I wanted to pull the cord in the first place). If I am walking out then I am usually halfway out the door and have to turn most of the way around to bring the cord within my field of view. Catching the cord sometimes requires moderate care and a few tries, but in the end I have never failed altogether to get the lights switched.

The dynamics that tend to mitigate trouble are many and varied. Together they render most trouble harmless. Future work should catalog them more thoroughly. In so doing it can characterize the kinds of trouble that *don't* tend to take care of themselves. Rather than pursue the dynamics of trouble in premature detail, let us summarize some of the more obvious points.

- Trouble is fatal vanishingly often.

- Most trouble isn't even significantly troublesome.
- Nothing works the first time, but experience is an efficient teacher.
- Much trouble couldn't be anticipated if you tried.
- Cultures go out of their way to warn you of hard-to-anticipate trouble.
- When they don't, people get killed.

In short, much trouble doesn't need to be anticipated and much trouble cannot be anticipated. What does this mean for planning? We have been seeing that it won't do to understand planning as Planning, trusting one's Plans to an executive that doesn't know what it is doing. On the other hand, there is no doubt that people regularly make and use plans. But such plan-using activity is the exception; the rule is a background of moment-to-moment improvisation. Matters that stand out against this background as problematic might become subject to the more sophisticated dynamics of explicit planning-ahead. I don't pretend to understand these dynamics. (See Agre and Chapman 1988 for further discussion.) The point is that plan-use is an advanced topic, something we shouldn't expect to understand until we understand quite a lot about the dynamics of moment-to-moment improvisation. (Some useful computational ideas about trouble and anticipation have come out of the Yale school. See especially Hammond 1986.)

One way of understanding improvisation is to start once again from Planning. When trouble arises, it is critical to know what you're doing. When a dumb executive runs into trouble, it can only hand control back to the Planner, which is supposed to know what is going on. Imagine, though, the plight of a Planner suddenly awakened to discover spilled milk, skidding tires, burnt fingers, angry people, or dented furniture. It has little clue how any of this came about since the executive was blindly following orders when it should have been keeping its eyes open. It has to reconstruct what it was trying to do, interpret the newfound damage or danger, assess its consequences for the ongoing project, and make a new Plan that resolves the trouble and gets the project back on the road. All of this would now be a lot easier if it had been on top of the situation all along.

If an agent is split into a Planner and an executive, the only time it knows what it is doing is when the executive performs the very first action specified by its Plan. The Planner has presumably arrived at its Plan through some definite line of reasoning, starting from its assessment of the situation at that moment. Let us speak of this line of reasoning as an *argument*, returning to what *arguing* means later on. Assuming the world didn't change significantly while the Planner was at work, the agent now has an argument for taking *that* action in *that* situation. Suppose the executive performs the Planned actions without incident and returns control to the Planner. In a busy or uncertain environment, the Plan was unlikely to have been very long: a few seconds at most. Neither the world nor the agent's goals are likely to have changed drastically in that time. A mislaid spoon might have forced re-Planning, but the dinner ingredients are still there and an hour of dinner preparation remains. If we compare the arguments leading to the old and new Plans, they will be almost identical. If your sauce needs stir-

ring, you haven't got the time to rederive the whole argument from scratch. Interleaved Planning needs to be performed incrementally. This is where dependencies come in.

We saw in Chapter B2 how an agent can use dependencies to incrementally update its reasoning. When a Planner makes a Plan, it should record justifications for all its arguments. It is useful to think of the argument motivating the agent's current actions as an object, the *argument structure*, which the agent incrementally modifies. When the agent's premises change, perhaps because it has noticed a change in some aspect of the world, it only has to rethink the parts of the argument structure that depended on the aspect that changed. Sometimes changes in the world call for drastic changes in the argument structure motivating the agent's actions. If the agent is starting to make dinner, it might notice a fire or a dinner invitation and change course completely. These things don't happen very often. The change is almost always very small. As we will see, the agent's argument structure is not a datastructure inside a machine. Like almost everything else in this theory, it is a dynamic fact, an epiphenomenal property of the agent's activity in the world.

Recall the definition of a running argument:

> A running argument is a continually incrementally updated hierarchical argument about what to do now.

We can now define a few of these words. Other things being equal, shorter increments between Planning steps are better. At each Planning step, the agent can take account of contingencies and opportunities. Short increments mean that the agent need only plan the aspects of its activity that cannot be improvised. One might have 10 increments per second, or 100, or 1000. As we take this process to its limit as the increment length goes to zero, we get a running argument. Running arguments do not bother separating Planning and execution at all. The agent's perceptual systems continually update its premises; the continually evolving premises support a continually evolving argument structure. Whatever action is justified by the agent's current argument structure is automatically performed. The arguments motivating a moment's action might refer to plans, laws, or heuristics, but above all they will refer to the agent's current situation.

Section B4e will make the idea of continual incremental updating more concrete. First, though, we have a couple more words to discuss, hierarchical and argument.

B4c Hierarchy as datastructure vs dynamic

In AI research, it is commonly held that human action is hierarchical in nature. Though this claim certainly has something to it, this section will argue that its usual interpretations are mistaken. After a brief tour of the vast metaphorical territory covered by the word 'hierarchy', I will introduce the distinctions we will need to discuss the computational issues. These are:

- Two sets of metaphors used, sometimes together, in talking about hierarchies. There are images of levels, some higher than others. There are also images of

trees, with roots and terminals and branching points in-between. AI tends toward tree-images.

- Two very different ways in which something might be hierarchical, by convention or by design. A human organization might be hierarchical, but only as a commonly-believed-in fiction. Engineered artifacts, by contrast, might be hierarchical through having been designed that way.

- Two views of hierarchy, the recursive-decomposition view and the metabolic view. Recursive decomposition is familiar from notions of subgoals and subroutines, each of which is defined by the has a 'postcondition' or 'output' it produces upon completion. Metabolism concerns the involvement of an agent in the cyclic patterns of its physical and social world. Neither of these views is right or wrong and others certainly exist.

- Two ways of understanding the recursive-decomposition view, as concerning machinery or dynamics. Both are possible and compatible views, but I will argue that only the dynamic view is correct. (The metabolic view only makes sense in reference to dynamics.)

The rest of the section discusses the dynamics of hierarchically organized activity. My thesis is that the 'simplifications' implicit in most conventional AI views of hierarchy make things harder than they have to be. Viewing hierarchy as a dynamic phenomenon instead of a property of datastructures suggests new ways of answering some longstanding questions.

The word 'hierarchy' ranges over a loosely-knit family of related metaphors. All involve some notion of some things being *higher* than others. The family of hierarchy-metaphors might be arranged into two overlapping clans. One of these invokes an image of *levels*, some of which are higher than others. The other invokes an image of a *tree* (as a diagram, not a real tree), with each member a vertex. Often there is a general lattice instead of a tree, though this distinction is not always clear. Either image or both can be present. The levels-image tends to emphasize the relationship of adjacent levels (which may or may not be different in kind); the tree-image tends to emphasize individuals and their relationships. Often (as in Chapter 1 of Miller, Galanter, and Pribram 1960) the applicability of the level-image is inappropriately taken to imply that of the tree-image.

Technical ideas about computation have used a wide range of hierarchy-ideas, but particular images predominate. Tree-images are usually stronger than level-images. Though one often speaks of levels, these are usually poorly defined except among the topmost and bottommost members. Often the word 'tree' actually names a lattice. The members of a hierarchy tend to be hardware components, hunks of program code, datastructures, or mathematical abstractions directly related to one of these. The notion of hierarchy is usually bound up with the much stronger notion of modularity (*cf.* Simon 1970). Metaphorically, modularity suggests that members do not 'know about' the

members above them. Members relate as parts and wholes. Each individual is self-sufficient, offering a definite contract to all takers.

A computer 'is' hierarchical in a different way from, say, a business. The hierarchical organization of a given computer (given accepted design practice) is practically a law of nature, entirely governing the flow of control and data in the machine. By contrast, the hierarchical organizational charts of a business only approximate the actual interactions and power relationships among its members. The hierarchy is prescriptive, not descriptive. It 'governs' the organization in an administrative sense, not as a natural law. Members presumably 'believe in' the charts, but the charts only retain their status as accepted fictions as a continual accomplishment of those members (*cf.* Heritage 1984, especially Chapter 7).

The distinction between these two forms of being-hierarchical becomes critical when we investigate the hierarchical ideas that people use to help organize their activity. Obviously, people regularly consult hierarchical representations like outlines or charts for guidance in finding information, interpreting orders, organizing searches, or anticipating the actions of others. But talking about someone's hierarchy-use is quite different from talking about hierarchical aspects of someone's activity. Many of the lessons we drew about plan-use apply to hierarchy-use as well:

- Neither plans nor hierarchies directly generate anyone's activity.
- Hierarchies, like plans, are resources that people use in situations.
- Neither following a plan nor using a hierarchy is governed by any fixed procedure. People regularly rearrange, reinterpret, interleave, interpolate into, adapt, and abandon their plans. Likewise, operations like establishing and amending hierarchies are regularly judgement calls, amenable to no fixed procedure and susceptible to influence by arbitrary details of the situation.
- Activity informed by a plan may or may not involve considerable improvisation. Activity informed by a hierarchy may or may not have an analogously hierarchical form.
- Regularity in one's activity may or may not be due to plans one is following. The hierarchical organization in one's activity may or may not be due to hierarchical representations.

The dynamics of hierarchy-using, like the dynamics of plan-using, are a special case of the dynamics of running arguments, an advanced topic. First we need to understand the basic dynamics of running arguments. This section simply concerns the hierarchical aspects of these dynamics.

Our question, then, concerns machinery and dynamics. Suppose we regularly observe people acting *as if* they were following a hierarchically organized plan. We would quickly conclude that there are hierarchical aspects to the dynamics of everyday activity. But we

wouldn't have grounds to conclude that there are hierarchical aspects to our machinery. As we will see, there are scenarios in which an agent with nonhierarchical machinery would end up engaging in hierarchically organized activity. What forms of dynamic hierarchy *need* to be explained by hierarchical plans and the like? Under what conditions do these occur? To answer these questions, let us compare two views of hierarchy, the *recursive decomposition* view and the *metabolic* view.

AI research has generally formulated hierarchy in activity as a *recursive decomposition.* (The notion originates with GPS (Newell, Shaw, and Simon 1960). Miller, Galanter, and Pribram (1960) took the idea from GPS and incorporated it into their definition of Planning.) Recursive decomposition envisions a tree or lattice structure in which every nonterminal element has been decomposed into parts and the terminal elements are drawn from a fixed set known as *primitives.*

Recursive decomposition originated in software engineering, particularly in the programming style encouraged by languages like Lisp, where a large number of small 'procedures' 'call' 'subprocedures', with the actual work getting done by 'primitive procedures'. (Note that I have appropriated the also-common term 'routine' for my own purposes. See Chapter A3 and SectionB2e.) In the early 1970's, Planning research explicitly borrowed software ideas, modelling Plans after procedures and Planning after programming (Sussman 1975, Waldinger 1975). On this view, one uses a Plan to pursue a goal, perhaps invoking sub-Plans to pursue subgoals, and eventually invoking 'primitive actions' that cause change in the world (Sacerdoti 1977, McDermott 1977). Here is a hierarchical decomposition of the sort one would expect an AI Planner to produce:

```
make breakfast
  assemble materials
    get out utensils
    get out cereal box
    get out milk
    get out OJ carton
  combine materials
    make cereal
      open cereal box
      pour cereal into bowl
      open milk
      pour milk into bowl
    make OJ
      open OJ carton
      pour OJ into glass
  consume food
    repeat: eat a spoonful or drink some OJ
  clean up
    bring utensils to sink
    wash utensils
  put away
    put cereal away
    put milk away
```

```
put OJ carton away
put utensils away
```

This Plan decomposes top-level goal of making breakfast into five subgoals: assembling materials, combining materials, consuming the food, cleaning up, and putting away. Each subgoal is further decomposed into subgoals of its own. Down at the bottom of this tree, there will be primitive actions, which might be at the level of picking up an object. (Although many AI Planners have had the capability of choosing among a number of different decompositions of a goal, it is exceedingly uncommon, as far as I can tell, for a Planner to actually have more than one way of decomposing any of its goals.)

According to this account of Planning, one's activity is hierarchical insofar as one's Plans are hierarchical. This view is coherent, but it is mistaken. It is the sort of explanation that results from inattention to dynamics: lacking a worked-out idea of how hierarchical patterns might arise in an agent's interaction with the world, it ascribes dynamic hierarchy to hierarchy in machinery. A single version of hierarchy, the recursive-decomposition view, has gotten a monopoly. Certainly the metaphor of goal trees provides one useful perspective. Lacking competition, though, it has become cast in concrete as the metaphor we use to answer every technical question we ask. We can see the effects of this bias by considering a competing view of the hierarchy in everyday activity. Observing the constructive interference of the two views can help us develop technical ideas that go beyond simple literal mechanization of either view.

Let us call this competing view of hierarchy the *metabolic view.* Whereas the recursive-decomposition view emphasizes the pursuit of some definite end, the metabolic view emphasizes the continual cycles of reproduction that dominate the more routine parts of life. The cycles form a hierarchy insofar as some of them contain others. Every culture has its own cycles: there are cycles of planting and harvesting, the life cycles of individuals, the cycle of formation of families, ritual cycles over the week and year, the cyclic repetition of the day's schedule, respiratory and cardiac cycles, menstrual cycles, the growing and cutting of hair and nails, gait cycles for crawling, walking, and running, and the cycle of seasons with its attendant cycles of temperature, wind, water (rainfall, runoff, and well levels). Most work has cycles of its own, from harvesting one rice plant after another to filling in one form after another. Many everyday tasks are cyclic as well: chopping, stirring, sweeping, sorting, digging. These cycles form a hierarchy: one walks to fetch water, works a whole day most days of the year, and completes many years' work to see a new generation rise.

Furthermore, every household has an elaborate metabolism of its own. A hundred different supplies are used up and need to be replenished: soap, clean clothes, gasoline, nails, postage stamps, subway tokens, bandages, coffee, garbage bags. Staple foods have different cycles according to their shelf life and the storage space they occupy: one might get fresh milk and bread every couple days, fresh meat and vegetables every week, fresh coffee and rice every couple weeks, and fresh salt and sugar every couple months. Garbage accumulates and is disposed of. A thousand different items of equipment wear

out and need to be replaced: batteries, cars, clothes, shoes, tires, motor oil, record-player needles. (Others, like coffee mugs, pens, and clothespins, vanish mysteriously.) Some have regular maintenance schedules; others are restored to working order in response to their regular wearing down (dull knives are one example, though many other items have become disposable in our culture, at least for most people). Every day's household chores combine episodes in some set of household cycles.

An AI metaphor gets AI into trouble in exactly the areas it de-emphasizes. If we use a single metaphor to understand everyday life, we will never notice the parts of life to which our metaphor doesn't apply well. Thus, in comparing the recursive-decomposition and metabolism views of hierarchy, the point is not to decide which one is more accurate. Instead, let us observe how the emphases of each metaphor remind us of the other's blind spots.

- Recursive decomposition describes a single episode. You decide to make breakfast, you unfold your breakfast-plan, and you're done. A goal has been achieved. Metabolism takes a longer view, surveying an eternal cyclic routine. Everything you accomplish will need to be accomplished again.

- Recursive decomposition evokes the possibility of production and doing something new. Metabolism evokes the necessity of reproduction and doing the same things repeatedly.

- Recursive decomposition suggests viewing yourself as essentially independent of the outside world. Your choice of which way to decompose each goal might be strongly constrained by extraneous considerations, but these aren't systematically articulated. Metabolism suggests viewing yourself as part of the world and emphasizes the network of everyday practicalities you live within.

- Recursive decomposition assumes a sharp focus, one goal at a time. Metabolism continually locates you at the intersection of a hundred cycles and poses the problem of continually negotiating all their claims. The dynamics of this negotiation are fascinating. Some cycles tend to be in phase, others out of phase. Every cycle assigns chores; cultures set up ways of living that keep the chores manageably distributed.

- Recursive decomposition must be supplied with a goal from the outside. The Planner may know why it is pouring the milk, but it doesn't know why it is ultimately making breakfast. There is an unexplained First Goal. Metabolism supplies a steady stream of reasons in the continual furthering of life. Everything that happens and everything that needs doing takes its place in a cycle.

- Recursive decomposition envisions nested activities reflecting nested goals (though see Allen and Koomen 1983). Metabolism reminds us that activities overlap. It also reminds us that many motivations aren't goals because they are never really

achieved. (Some AI programs have contained 'goals' like 'staying alive'. This is worrisome. It stretches the word beyond recognition.)

It is tempting to take conceptual revenge by celebrating the holist virtues of the metabolic view and talking down the reductionist vices of the recursive decomposition view. But this would be a mistake, and my argument does not require it. One need not adopt a single unified account of the role of hierarchy in everyday life. The tension between these two metaphorical systems is a benefit, challenging us not to accept a superficial stock account of any specific dynamic issue.

Many dynamic phenomena can coexist, but there is only one architecture. Adopting only the recursive-decomposition view of hierarchy suggests equipping the architecture with machinery for assembling and manipulating tree-shaped structures. Adopting only the metabolism view of hierarchy would suggest equipping the architecture with machinery for assembling and manipulating cycle-shaped structures. Of course, we could always postulate both sorts of machinery, and twenty others as well, and then ask how they all work together. This would not be very parsimonious. Tolerating both views of hierarchy should make us step back and consider the large and difficult question of how the interactions between an agent and the world could have both sorts of patterns.

An agent might have a collection of ways to decompose goals into subgoals. Let us consider two extreme approaches the agent might adopt. The first approach is to make a complete Plan. Given a goal, the agent assembles an exhaustive goal tree by decomposing every subgoal all the way down to primitive actions. The second approach is to delay decomposing each goal as long as possible. Given a goal, the agent decides on a set of subgoals, chooses one to pursue first, sets itself that single subgoal, decomposes that subgoal into subsubgoals, chooses one of them, and so forth. The arguments that enter into the agent's choices might make reference to the future, but the agent only ever decides what to do now. Thus the agent performs the first primitive action before it even decides what the second subgoal is going to be at any level. The agent never performs a decomposition or imposes any ordering on subgoals before it has to.

Everything being equal, these two extreme approaches will result in exactly the same sequence of primitive actions being performed. In each case, the agent's activity will look like a tree, a recursive decomposition of the original goal. In the extreme Planning case, this tree will result from the agent's executive reading off an identical tree inside its head. In the extreme improvisation case, the agent will never have decided anything except which subgoal of each goal to pursue right now. The two extremes define a trade-off. Making choices immediately requires complex reasoning but it anticipate later problems. Postponing choices means more information will be available but it risks painting oneself into a corner. (Although Miller, Galanter, and Pribram explicitly discuss the possibility of postponing the expansion of Plan hierarchies (1960 p. 16), the first system to postpone decomposition decisions in a general way was NASL (McDermott 1977). Along these lines, Wilensky (1983) discusses the specific technique of postponing the decomposition of subgoals until the last minute by effectively making the Planner and executive mutually recursive functions of the Plan's decomposition. Sepa-

rate provision must be made for the possibility that the agent might wish to abandon already-made decomposition decisions in the face of unforeseen conditions.)

In an unforgiving and predictable world (an odd combination found only in board games), the case for building detailed goal trees ahead of time is strong. In a relatively benign and unpredictable world, the case for goal trees is weak. The case for goal trees is particularly weak if decisions about decomposition and ordering can benefit from past experience. If the agent must decide correctly every time it needs to perform complex simulations to anticipate troubles. If it has time to learn from experience then this need is lessened.

> The kitchen sink was broken, so I had to wash out my tea cup and tea infuser in the bathroom sink. My hands, already informed by much past experience of washing out tea cups, improvised the following sequence. When I entered the bathroom, the cup was in my right hand with the infuser in it. As the sink approached, I passed the cup to my left hand, removed the infuser from it with my right hand using a grip that would then let me grab the cold water knob, reached for the knob with my right hand while placing the cup under the tap, turned the tap on, rinsed out the cup while removing my right hand from the knob, laid the cup aside while adjusting my right hand's grip on the infuser, rinsed out the infuser under the still-running water, withdrew and closed the infuser, and ... observed that the sink was full of ugly tea leaves. Grudgingly deciding it'd be polite to rinse these down the drain, I picked up the tea cup, let it fill with water, dumped it around the outside of the sink, and ... realized I'd already done this a moment ago. (The dynamics of this amazing sequence are interesting in themselves, but they are beside the present point.) When I next returned to wash out my cup, I went through the same sequence, only to pull up short as I was about to rinse my cup. Reminded of my conclusion from the previous episode, I first rinsed the infuser instead. This required an extra couple hand motions which I found myself repairing the third time.

In this story an ordering decision led to duplicated work. Taking this into account the next time led me to reverse the ordering. A proper analysis of this story would require a more thorough description of its dynamic context. Since I usually drink one cup of tea after another whenever I am working, this would involve an analysis of the cycles of making cups of tea, cleaning up, keeping tea in stock, and caffeine addiction. Tea-making expeditions are usually combined with other chores, linking tea-making with other dynamic systems. Further, many of the steps in the tea-making cycle are also parts of other activities, so we should expect connections to their dynamics as well.

Here is a more complicated example:

> One week I was writing an article in the lab of some friends, in whose kitchen hot water could only be made by boiling it in a kettle on an electric stove. Decomposing the process of making tea in the obvious manner, I

(1) put a tea bag in a cup,
(2) poured hot water into the cup,
(3) waited a few minutes for the tea to steep, and
(4) threw out the used tea bag.

> In particular, I put the water on to boil after getting out the cup and tea bag. Annoyed by all the time it took waiting for the water to boil, I had plenty of opportunity to reflect on what I might have been doing instead of waiting. After a few times I finally realized that I could have been doing (1). Next time, as I went to get out the cup, this argument came to me and I put the kettle on to boil first. Shortly afterward, I further decomposed the substep of preparing hot water and realized that it is best to turn on the electric stove before filling the kettle rather than after.

Boiling water was part of step (2) of the decomposition because hot water was the second ingredient in tea. Having to sit waiting made me reflect on the wisdom of this decomposition. My new decomposition did not arise by magic; finding myself waiting made me explicitly wonder what I might be doing instead. The decision to decompose making-tea into steps (1) and (2) was part of the reasoning behind boiling the water, so I asked myself if I was doing the right thing. All this wondering involved several dynamic phenomena outside the scope of this report, including some subvocalized language. The point is that it worked.

One of the principal virtues of decomposing your goals as you go along is that you can change course as circumstances warrant. At any given time your current primitive action will be informed by goals at several levels, for example, a chain of subgoals like making breakfast, assembling materials, fetching the milk, opening the refrigerator, reaching for the refrigerator door handle, and raising your arm, where each goal is a subgoal of the one before it. (Obviously this is a cartoon goal chain. Later chapters will discuss the nature and content of goals in more detail.) As you raise your arm, you might spot the milk sitting on the kitchen shelf. If you are awake, you will realize that fetching-the-milk is no longer grounds for opening-the-refrigerator. Lacking any other reason to open the refrigerator, you will presumably want to drop that goal and the others below it. You might then adopt a new subgoal of fetching-the-milk, such as approaching the region of shelf where you've noticed it. All this is greatly simplified if you have had the foresight to keep dependencies on your reasons for adopting all your goals. Once your decision to pursue a goal loses its support in the dependency network, you should stop pursuing it.

In general, you might decide to drop any subgoal at any time. In this sense, subgoals are *ephemeral*, merely means to an end. Since every goal is a subgoal (unless there is an ultimate First Goal), we must conclude that *all* goals are ephemeral. It is best to think of goals, whatever they are, as tools. A goal is not so much part of you as an object you use, much as you would use a recipe or a map. There are unlikely to be simple, fixed, mechanical methods for using goals, any more than for using plans or hierarchies. In

particular, there are unlikely to be fixed, automatic methods for determining whether a goal has been achieved—much less specialized machinery implementing such methods. AI Planners have generally assumed that goal-achievement is a clear-cut matter, that it is always possible to judge whether goals have been achieved, that the fixed executive can be trusted to make these judgements, that these judgements do not require any Planning themselves, and so forth. (Soar is an exception due to its universal subgoaling mechanism; see Section B4d.) But if goal A is a subgoal of goal B, then a judgement of whether A has been achieved will depend on the context and, in particular, on B. To see this, think of any occasion when you'd speak of 'cutting corners' or when we might have the following conversation. You: 'Could you please accomplish A?'; Me: 'How about if I just do X?'; You: 'That's good enough'.

Ten ways in which the judgement of goal-achievement can depend on context:

- when computing a number, pi can be 3 for rough feasibility checks
- travelling to London might mean East London or all the way into town
- powering a light, a portable generator is OK if it's only for an evening
- clearing of weeds has to be more thorough near weaker plants
- giving directions, strangers require more details
- opening a window, how wide depends on what has to go through it
- adding salt, to taste
- choosing a object to use as a paperweight, your choice is subject to the other uses you might have for the various candidates
- fetching a bottle of wine for dinner, what's on hand is OK for everyday, but special occasions call for a trip to the wine shop
- painting a wall, one coat is OK in a closet, but the kitchen needs two

A Planner, or any other device that is given a goal to pursue by decomposition, is in a rough spot. Lacking any idea of *why* it is supposed to pursue this goal, it has no way to make all the judgement calls that go into normal goal-using. This is why Planners must assume that there are fixed, automatic methods for using goals. A Planner tries to work out of context.

This conclusion, though true as far as it goes, is not very helpful. It is notoriously difficult to give a general account of the ways in which context can influence judgements. Context quickly gets out of control, seeming to seep into every crack of our attempts at reasoning. The very word 'context' offers us the dangerous invitation to address the 'problem of context', as if there were a single object or a homogenous substance called 'context' to be explained. Faced with the problem of context, one has about four options.

- Ignore it and hope it goes away. Deride as mystical anybody who insists on its importance.
- Formalize exactly the bits of it you need for each specific example without addressing the question of how to formalize it in general.

- Bluff by introducing a formal object called 'context' (or 'situation'), passing the whole voluminous package as an extra argument to your Planner or semantic theory.
- Spell out its bizarre properties and use indiscriminate appeals to holism to celebrate it as the solution to all problems.

None of these moves is very helpful. The phenomenon of context is better viewed as a symptom of a deeper problem. Something like 'context' will pester any ill-considered partition of research into human activity. Dividing the organization of everyday activity into 'Planning' and 'everything else' was a nice try, but the 'problem of context' is trying to tell us that 'everything else' is refusing to break cleanly away. The idea of Planning treats non-Planning phenomena like improvisation as marginal, but (as it turns out) they *aren't* marginal. Context is the leftover trace of this repressed margin. Like any repressed problem, it festers, manifesting itself in innumerable ways, each of which appears as a technical difficulty. Among these difficulties are the recalcitrant inflexibilities of Planning, such as the seeming need for fixed methods for detecting goal-achievement. No doubt one could extend Planning to patch this particular manifestation or any other, but only at the price of another one appearing somewhere else.

The problem of context is one more indication that our machinery does not directly manipulate plans, trees, and goals. Running arguments—continual incremental updating of the argument motivating action—are an account of what the machinery *does* do directly. Nobody could deny that plans and goals exist (though trees are more questionable). Using plans and goals is one thing a running argument might do—one of the more difficult things. Interposing running arguments between the machinery and the plans and goals is not simply a matter of reconstructing the latter at a 'higher level of abstraction'. It makes an important qualitative difference. Neither plans nor goals work according to fixed rules or fixed procedures. People can decide to use plans and goals however they wish, according to custom, perversity, or the demands of the situation. The task now is to inquire into the dynamics of when and how people use plans and goals.

The aim of this this long section has been to shake loose some customary ways of thinking about hierarchically organized activity. It has not tried to provide any finished answers. The question of the First Goal is still a serious problem (named, by the way, by analogy to the medieval problem of First Cause). Is there really a definite goal at the top of everybody's recursive decomposition? What would it be? Survival? Love? Continuation of the species? Continuation of one's own genes? A better idea would be to replace the idea of a goal with a better worked-out theory of motivation. The real story is presumably long and complicated. Rather than trivialize this difficulty, the demonstrations of Chapter B5 presume a particularly simple account of hierarchy, elaborating the straightforward recursive-decomposition view of running-argument dynamics.

B4d Argument and centralization

In speaking of a 'running argument', I am obviously using the word *argument* in some special way. My use of the word has only a loose and metaphorical relationship to its vernacular uses. I absolutely do not intend these 'arguments' as a model of real, live arguments, such as everyday quarrels or philosophical disputes. I will provide no formal definition of arguments, only a discipline of writing Life rules that fits with certain dynamic ideas.

Thus 'argumentation' here is a metaphor for a style of rule-language programming. What characterizes this style of programming is its other-things-being-equal control structure. Many rule languages have a deductive semantics, whereby the right hand side is a proposition to be unconditionally believed as soon as the rule manages to fire. Other rule languages have an imperative semantics, whereby the right hand side is an action to be unconditionally taken as soon as the rule manages to fire.

Section B4b, in discussing improvisation, sketched a few ideas about arguments. From moment to moment, it suggested, you conduct an argument with yourself. On each next moment, through your arguing, you arrive at some conclusion about what to do, and you do it. Naturally these conclusions will often seem premature, but you have to do *something*, if only to keep breathing and maintaining your balance until you can settle on something more elaborate.

Section B4b also referred to "argument structures." An argument structure evolves by constant incremental changes. These changes in an agent's argument structure ought to be small when the agent's situation is changing slowly and large only on the relatively rare moments when the agent's situation changes drastically. Argument structures might look like the diagrams logicians draw to trace how some conclusion might be justified from certain premises, but the arguments' coherence isn't enforced by any of the local, formal standards of admissible inference that govern systems of formal logic.

My ideas about arguments descend from those of (Doyle 1980). Doyle introduces the notion of "dialectical argumentation." Doyle's concern, unlike mine, was to make a model of reflective thought, that is, the sort of thing you do when you struggle with a big decision. Doyle assimilates thought to action. Deciding what you think is like deciding what to do. You decide what to do by conducting an argument within yourself. Any active component of oneself can offer suggestions about what to do and adduce arguments about why. When disagreements arise, choosing among the arguments is itself a decision to be made by argumentation, recursively: each component may adduce arguments as to why its arguments are better than the others.

Before going into detail, let's consider a silly cartoon example:

```
(propose (hold-up baybanks))
```

Contradictory arguments are put forward:

```
(propose (support (hold-up baybanks) money-in-it))
(propose (object (hold-up baybanks) it-would-be-wrong))
```

The first argument encounters no objections, so it is accepted:

```
(take (support (hold-up baybanks) money-in-it))
```

But some part of the system considers the second argument inferior and proposes that it be considered so.

```
(propose (object (object (hold-up baybanks) it-would-be-wrong)
                 prefer-practical-to-moral-arguments))
```

There being no objections to that line of argument, it is accepted:

```
(take (object (object (hold-up baybanks) it-would-be-wrong)
              prefer-practical-to-moral-arguments))
```

Consequently, the moral argument against robbing the bank is rejected:

```
(blocked (object (hold-up baybanks) it-would-be-wrong))
```

Since no other objections are outstanding, the motion stands:

```
(take (hold-up baybanks))
```

Dialectical argumentation, then, involves the adducing of arguments and counterarguments concerning a proposal for action. Because each argument and counterargument is itself an action, the argumentation process itself is fair game for argumentation. The system approaches the problem of deciding between conflicting arguments in just the same way that it approaches any problem in the world. The method has some important properties:

- All reasons for action are *defeasible*, meaning that they might be overridden if there are good reasons to do so.
- The decision process is *additive*, meaning that all parts of the system can contribute to the reasoning in a uniform way, by contributing arguments.
- Individual decision processes are automatically converted into dependency networks. Thus an arbitrarily complex argument structure can be used many times a second. In particular, an argument's conclusions will stay IN as long as its premises stay IN.
- Because an argument recorded in the dependency network will be recapitulated whenever its premises are satisfied, it will be automatically carried over to analogous future situations.
- The system's reasoning can be modified without ever introducing a side-effect. By simply providing an argument explaining why an action is a mistake, a person or program can arrange for it to be overridden in the future.

The rules in a running argument do not prescribe actions, they simply propose actions. One rule might propose that the agent lift its hand. Another might propose placing its hand on block B. Another might propose moving left. As all of this arguing gets converted into network stuff, we can imagine various regions of the network arguing with one another. One region might have proposed moving left and another might have proposed moving right. Since these proposals conflict, the one region might raise an objection to moving left, perhaps on the grounds that doing so would knock over the tower. The other might respond, you can't move right because getting to the objective by moving right would take too long. A third region of the network might then step in and object to the latter objection on the grounds that it is weaker than the former. In general, one can propose objections to objections to objections arbitrarily deeply. It is up to the programmer to make sure that the system neither deadlocks nor attempts to perform conflicting actions.

Despite all the anthropomorphism, all of this proposing and objecting is implemented by ordinary rules. The rule system itself does not have any special knowledge of proposals and objections; they are all just list structures in a database. To propose an action you assert:

```
(propose action)
```

To object to an action you propose objecting to the action on some grounds:

```
(propose (object action reason))
```

You don't simply object, you propose an objection. In the center of the action is a rule that says that any proposed action is taken unless some objection is sustained against it:

```
R1: (if (propose ?action)
        (unless (take (object ?action ?reason))
          (take ?action)))
```

This three line rule, R1, is the soul of argumentation. Notice that its UNLESS rule has an unbound variable, namely ?reason. Even if R1 fires once for a given binding of ?action, and even if that action actually gets taken, someday some new objection might come along to defeat the proposal. If this happens, the action will no longer be taken. If the action is currently under way, it will stop. If the proposal to perform the action is asserted again and the objection is still in force, the proposal will not be adopted. The new-found objection will amend the AND-NOT gate in the dependency network corresponding to this UNLESS rule, adding a new inverted input to the gate through the mechanism Chapter B2 called intervention.

Let us consider some examples of this rule in operation in the blocks world introduced in Section B3c. Suppose some rule proposes that the agent move its hand to the left:

```
A2: (propose (move hand left))
```

When this proposition comes IN, **R1** will fire, producing the following:

```
R3: (unless (take (object (move hand left) ?reason))
      (take (move hand left)))
```

In other words, move the hand left unless there is some reason not to. The UNLESS rule, **R3**, will generate a non-monotonic dependency, an AND-NOT gate. For the moment, this gate will have no inverted inputs since no objections have ever been raised to that proposal. The output of that gate will be:

```
A4: (take (move hand left))
```

But suppose that some rule files an objection:

```
A5: (propose (object (move hand left) (would bump tower)))
```

Now something more complicated happens. This being a proposal like any other, rule **R1** fires a second time, producing a rule that will accept this objection unless some objection is sustained against it in turn:

```
R6: (unless (take (object (object (move hand left)
                                  (would bump tower))
                          ?reason))
      (take (object (move hand left)
                    (would bump tower))))
```

(So far as the rule system is concerned, both the actions and the reasons are arbitrary list structures. It is up to the user to formulate a consistent ontology of actions, reasons, and so forth and to express these in a real representation. This example uses a toy representation for simplicity, though none of the system's representations are very convincing either.)

This new UNLESS rule, **R6**, now checks for second-order objections. If none are present, then the UNLESS rule will license its conclusion:

```
A7: (take (object (move hand left) (would bump tower))
```

This sustained objection will now attract the attention of the original UNLESS rule of a moment ago, **R3**. Thus the proposal of moving the hand left will not be adopted. As we mentioned, **R3** has most likely already fired, creating an AND-NOT gate whose output is **A4**. If so, then this gate will now receive an additional inverted input, namely **A7**. And the agent will not move its hand to the left.

The proposed action in this example, moving the hand left, happens to be one of the system's primitive actions. At the end of every clock cycle, the motor system decides whether to move left by checking whether **A4** is IN or OUT. These conventions about proposals and objections apply equally well to compound or abstract actions. In each case, though, when the system adopts some proposal, that adoption is only good for

the current clock cycle. If some compound action covers many clock cycles, it must be proposed, argued for, and adopted on every one of those cycles. This is not a great computational burden for the system so long as all this arguing, at least after the first cycle, takes place in the dependency network and not through the firing of new rules. In other words, the system takes its actions only so long as they are supported by argument.

Rule R1 is special in that it explicitly relates proposals and objections to actions. All the other rules propose actions, raise objections, and adduce reasons pro and con. In practice one programs with a collection of macros, all written as rules themselves, for expressing complex ideas about priorities, weighing of arguments, decomposing compound actions, and proposing alternative and interpolated actions. The details of these methods of argumentation are not especially original or general, nor do they bear on the theoretical issues at hand.

We could construe the notion of argumentation in two different ways, according to who has the burden of proof in putting forward their argument. A system could have an inner loop that says, "decide what to do then do it." That method poses the positive task of choosing some particular action to take, one action at a time. The running argument system takes a different, decentralized view. Any patch of dependency network can make proposals and the proposed actions are taken by default. Rather than having to positively argued for an action, you just propose jumping off the cliff, and if nobody offers any objections then you'll jump. All through the network proposals are being made, arguments are being conducted, objections and supporting evidence are being offered.

It would be nice to have some unified theory of argumentation, about evidence and its weighing and so forth. Philosophers have been looking for such a thing for millenia and I certainly haven't got one. The system now uses a mishmash of a couple different theories of argumentation that I tried out; it gets baroque. In any event the important principle is that proposed actions are taken by default.

The point of this decentralized style of programming is that rules do not have to offer guarantees. If a rule proposes some an action without ironclad guarantees that it is the optimal thing to do, then that information can be embodied in separate rules that raise objections in appropriate situations. (Compare Minsky (1980; 1986 Chapter 27) and Winston (1983) on censors.) The world is a complicated place; your rules can address prototypical cases and leave the endless enumeration of exceptional cases for an endless later. In practice, new rules get written when one observes the system making a mistake, going into a loop, floating off into space, or seizing up, one stops it. After looking at its arguments, one can formulate a new rule objecting to the erroneous proposal and perhaps proposing an alternative. (Recall the discussion of incremental repair of rule sets in Section B3g, where we extended the rules to cause the townspeople to conclude that the shepherd is a liar.) The system accumulated several dozen arguments about blocks world over a period of several months. I'd let friends watch the system; when they'd complain about it I'd try to convert their complaints into new rules. The system

still isn't very good about circumventing obstacles and working in tight spaces. But it does engage in interactions of some complexity, as we will see in the demonstrations.

Let us consider an example rule.

```
(if (and (propose (try (grasp ?x)))
         (on hand ?x)
         (on hand ?y))
  (if-shown (neq ?x ?y)
    (suggest (prefer-option
               (try (center-on hand ?x))
               (try (grasp ?x)))
             (avoid-unnecessary-grabbing ?y))))
```

This rule uses the **and** and **if-shown** constructs described in Section B3h. In English it says, "if we have proposed grasping **x** and the hand is on two different objects **x** and **y** then propose postponing the grasping operation until we have had a chance to center the hand on **x**." The **suggest** form combines making a proposal with proposing a reason to support it:

```
(if (suggest ?action ?reason)
  (propose ?action)
  (propose (support ?action ?reason)))
```

(Bear in mind that this is Life code, not Lisp code, so the semantics is not Lisp's if-then-else. An **if** rule with multiple consequences asserts all of them when it fires.) If nobody raises any objections to preferring centering to grabbing, then the system will adopt that preference. Taking this action will then offer support for centering and raise a concomitant objection to grabbing:

```
(if (take (prefer-option ?better-action ?worse-action))
  (propose (object ?worse-action
                   (preferable ?better-action)))
  (suggest ?better-action
           (preferable-to ?worse-action)))
```

This rule is part of the system's domain-independent knowledge about arguments. Observe how the argument about whether to take the action of grabbing **x** has spawned another argument about whether to prefer another action instead. Only once the system has resolved this second, inner argument can it resolve the first, outer argument.

Centering the hand isn't always a good idea, though. If block **x** already has a block on it, we shouldn't push it all the way off of **x** without a good reason.

```
(if (propose (support (prefer-option
                        (try (center-on hand ?x))
                        (try (grasp ?x)))
                      (avoid-unnecessary-grabbing ?y)))
```

```
(if-shown (and (on ?z ?x) (neq ?z hand))
  (propose (object (prefer-option
                     (try (center-on hand ?x))
                     (try (grasp ?x)))
                   (centering-would-shove ?z)))))
```

This rule is simpler than it looks. It reacts to the proposal made by the **suggest** form a couple rules back, checks whether something besides the hand is resting on **x**, and if so it objects to the proposal of preferring centering the hand on **x** to immediately grasping it. If no other arguments are put forward, this objection will cause the system to decline to prefer centering to grabbing. Lacking any other arguments, the system will go ahead with its original proposal of grabbing **x**.

Both Doyle's argumentation scheme and the one I have described have much in common with Laird's notion of *universal subgoaling* (1983). Laird, Rosenbloom, and Newell have demonstrated universal subgoaling in the context of the Soar architecture (1986). Section C5d will discuss Soar in detail. Universal subgoaling is, very simply, the idea that any decision an agent makes can become the topic of fully general reasoning. In the context of the Soar architecture, this "fully general reasoning" takes the form of search in a "problem space;" uncertainties about where the search should proceed lead to the spawning of a subgoal. This new subgoal itself becomes the object of a problem space, as if the system had called itself recursively (*cf.* Batali 1983 and 1985). Indeed, the system maintains a stack of active goals and only considers a single goal at a time. This is not as severe a restriction as it appears, since Soar has a scheme analogous to dependency maintenance, called "chunking," for summarizing problem solutions (see Section B5f). The difference between Soar's universal subgoaling and the running argument system's argumentation scheme is that the former is an explicit part of the architecture whereas the latter is a relatively informal programming convention. Universal subgoaling is thus a much better worked out idea than argumentation. At the same time, it carries considerable baggage. Is every activity organized by being aimed at a single goal? Section B4c has cast serious doubt on that view, but I will not offer a serious alternative in this report.

The remainder of this section is optional.

The system's arguments do get considerably more complex than these, but not in any theoretically important way. In particular, I tried to write a very general facility for weighing arguments for and against conflicting proposals. In retrospect this was a quixotic enterprise insofar as a system can't be truly decentralized if it has more than one way of doing anything. Suppose the system proposes two conflicting actions, A and B. Then Joe might see the proposal of A and, not knowing anything about B, propose an objection against A. At the same time, Jane might see B and, ignorant of A, offer support for B. And then José might see both A and B and offer support for the proposition that A is better than B.

Now we have three items of information about A and B, but they are all in different places. Some rule has to go around and make José's argument into support for A

and an objection to B. Or somebody has to make Joe's objection against A into an objection against A's being better than B and also make Jane's support for B into another objection against A's being better than B. Once the arguments have been made commensurable by having been gathered together as objections to or support for the same proposal, we can imagine some rules (general or specific) weighing them and arriving at a conclusion.

As a result I have some baroque rules for shuffling arguments all around to try making them commensurable. The Joe/Jane/José example is about the simplest; it can get much worse. What is worse yet is that it is hard to keep the system from redundantly arguing the whole thing through in 2 or 4 or 16 different ways courtesy of the symmetries in these generic rules.

In practice this is all nonsense. Life rule language programming may not involve serial execution, but most of the principles of good programming practice remain. When you write a line of code you have to know what other lines of code it might interact with. You have to use abstraction properly so you can formulate proper conventions about where to hang your arguments.

B4e Running arguments

There is no way to provide a precise, technical definition of the dynamics of running arguments; they will vary depending on the sort of activity you are engaged in. Instead, this section presents some useful ways of thinking about the dynamics of a running argument in particular situations. First, let us summarize the pieces we have already assembled.

1. Your architecture is divided into peripheral systems and a central system. The peripheral systems consist of innately wired modules concerned with early perceptual processing and with low-level details of motor control. The central system, so far as I will specify it, consists only of a dependency network whose inputs and outputs are connected to the periphery, together with a device called the 'source'.

2. Your central system starts life with nothing (or almost nothing) in it. Over the course of your life, a dependency network grows as the 'source' adds new propositions and justifications. I am leaving the details of the 'source' largely mysterious except to say that it is always concerned with the situation at hand, that it adds new connections relatively infrequently, that it is as simple as it can possibly be, and that it is probably very simple indeed. Thinking of the network as a *dependency* network rather than just an arbitrary combinational circuit will prove to be a big help in understanding its structure and dynamics.

3. Thus your dependency network reflects your past history of dealing with slightly novel situations. Once new elements are added to the network they never go away. At any given time, almost all the action in your central system is propagation of

values in the dependency network. Since the network's premises are the inputs from the periphery, values change in the network when these inputs change. Since the network's outputs drive the periphery's motor-control functions, the motor functions you are performing change when these outputs change.

4. At any given moment, a snapshot of your dependency network will assign values of IN (or 1) to some propositions and OUT (or 0) to the rest. This assignment changes as a continuous function of time. At any given moment, the set of IN propositions can be interpreted as an argument that you should take certain actions now. Considered as a function of time, this set of IN propositions is called the argument structure. The argument structure almost always changes very slightly because it is rare to suddenly drastically change what you are doing.

5. The argument structure is hierarchical in some way. Lacking a better worked-out dynamic account of hierarchy, I will assume each moment's argument reflects a recursive decomposition of some goal. Thus any given moment's action will be justified by a chain of subgoals successively decomposing a First Goal into one or more primitive actions. I will draw some pictures to help in thinking about this.

Section B4d concerned a process of argumentation that takes place over a single clock cycle. More interesting is the way in which the argument structure evolves over time as the agent interactions with its world. A running argument is just a continually evolving argument structure. Proposals and objections and reasons come and go over time. The argument structure 'evolves' in that little of it typically changes from one clock cycle to the next. A new objection might arise as the agent encounters an unexpected condition, but then it might be overruled, leaving the agent to carry on as before. An opportunity might arise to pursue some subgoal in a straightforward way, leading to a proposal which is adopted and acted upon and then which evaporates once the subgoal has been achieved. Every once in a while there comes a big change, such as in an emergency or when the agent finishes with one large subgoal and moves on to another. But usually nothing at all changes past the finest details of perception and motor control. In general, the argument structure changes to the extent that something different is happening. At least that's the ideal.

The best way to think about the evolving argument structure is to imagine the agent's whole combinational logic circuit spread out across a large sheet of paper, much as we did at the beginning of the chapter. Imagine a sort of brain scan, whereby the currently active regions of the network glow while the remainder stays dark. As the agent's interaction with the world proceeds, various regions will be lit at various times. Some regions will be almost always lit, perhaps indicating long-term projects or eternal truths. Other regions will only be lit in very exceptional conditions. And some regions will pulse in and out with fair regularity as the system goes through cyclic activities. Some regions will be relatively domain-specific, others might relate to forms of reasoning that come up occasionally in any domain, and others might relate to common bodily

motions. Perhaps some of it will never light up. And I imagine at least one node will light up whenever you see your grandmother.

Combinational logic is such grade-school stuff that it is easy to lose track of how profound it is. You have this large sheet of circuitry, with inputs on the left and outputs on the right, and all of it is running all the time. All of it is working to maintain some relationship between inputs and outputs. Every gate is always ready to change its state instantly if its output ought to have a different value. If every single gate is the equivalent of one rule, or as one instance of a rule, then a fantastic amount of computation is effectively happening all the time. An agent conducting a running argument does not take an action because some program counter reads 7. It acts because it has thought through what it ought to be doing. It knows what it is doing. A planner, by contrast, is something much shallower. Suppose the executive is moving the agent's hand to the left. The agent once understood why it should be moving its hand left, but that understanding is off in another module if it was saved at all. A system based on combinational logic will take unexpected conditions in stride, meeting the world's prevailing contingency halfway instead of attempting to rigidly impose its own will. If the agent has adopted a subgoal, it knows why, so that if that subgoal is no longer a useful means to an end it is no big production to abandon it.

Reid Simmons' thesis (1988a and 1988b) is about keeping dependencies on the Planning process so as to incrementally redo it in different situations. The innovation here is that the dependencies are always running the show, with no distinction between Planning phase and execution phase. Even if the rules make plans, it is the reasoning leading the agent to adopt a plan that runs the show, not the plan itself. That is important, for all the reasons we have discussed.

The notion of a running argument also offers us an account of the slogan of continually redeciding what to do. The system accepts a new set of perceptual inputs and delivers a new set of perceptual outputs in real time. That's still discrete, but it's a decent approximation to continual. The system sometimes needs to run some rules, but only the rules that are actually novel. If the agent's activity, once it has settled into a pattern, is almost entirely routine, then it shouldn't have to run very many rules. The slogan is, five rules per cycle. The scheme has its faults, which we will discuss later, but it provides at least one technical rendering of the slogans I posted as our goal at the beginning.

The hardest thing to accept about running arguments is that they are an account of the steady state dynamics of an agent's activity in a world. They are not an account of either the causes or effects of new thoughts. In terms of Chapter B2's vocabulary for dependencies, running arguments concern the dependency network, not the source. This can be hard to accept because we in AI are so accustomed to assuming that no activity is interesting unless it involves large hunks of novel thinking, whether by Planning or Problem Solving or Learning. This attitude is ingrained in our language. If I propose to address 'making breakfast', I will be understood to mean something like, 'making breakfast, never having made breakfast before, doing it without any help, and getting

it right the first time'. But I mean nothing of the sort. I mean 'making breakfast, given that you have done it enough that you can do it routinely'. Given the customary distinction between an impossibly smart Planner and a hopelessly stupid executive, it is hard to imagine how this can be an interesting topic for study. If the agent isn't thinking anything new, goes this argument, it must simply be executing a compiled Plan. This argument poses a double bind, demanding a choice between two equally unworkable alternatives. There is no way that one could make breakfast (a real breakfast, in a real kitchen) by executing a Plan. The whole point of a running argument is that it is vastly more flexible than executing a Plan. A running argument improvises rather than always trying to anticipate, takes advantage of opportunities rather than ploughing blindly onward, and responds sensibly to contingencies rather than circumventing them all through exhaustive conservatism.

Chapter B5

Experiments with running arguments

B5a Introduction

This chapter demonstrates the running arguments system. I built this system in 1984 and 1985 to help me think about the ways in which novel patterns of activity become routine. The system achieves a number of striking successes that distinguish it from previous architectures for artificial agents. Rather than basing its decisions about action on a shallow representation constructed by a Plan-construction module, the system effectively figures out on every cycle of its clock what to do now. The system integrates a traditional rule language with a dependency system that permits the system to accelerate its own operation without thereby complicating its reasoning. The mechanism that maintains the dependencies accelerates the system's operation gracefully and incrementally and is completely transparent to the rules' author.

After much effort, though, I concluded that the system has a couple of severe shortcomings. Extended reflection on these shortcomings led to the novel ideas about perception and representation that Part C describes and that were later embodied in the Pengi system. The system is worth discussing at length, not despite of its shortcomings, but indeed because of them. I built the running arguments system in the way I was taught to build AI systems. Its shortcomings and their consequences are shared by many other AI systems. Briefly put, these shortcomings concern the objective ontology of the system's representation of its world. This ontology has two principal consequences: the unrealistic demand that the perceptual systems maintain a world model and an account of abstraction that cannot be implemented without excessive computational effort and unnecessary architectural complexity.

Although these lessons seem obvious to me in retrospect, it is also clear in retrospect that many factors colluded to obstruct their discovery. The most frustratingly gratuitous of these factors is the misleading nature of 'blocks world'. Blocks world both hid these difficulties by tending to excuse unacceptable compromises and hindered their resolution

by suppressing the properties of real activity that permit the correct solutions to work.

This chapter has six sections.

Section B5b concerns the architecture of the running argument system and explains the technical accounts it offers of the ideas that motivate it. The system's goal is to do technical justice to the slogans of "knowing what you're doing," "continually redeciding what to do," and the "mostly routine" nature of activity. Chapter B3 has already gone into some detail about the relationship between the rule language and the dependency system, so this chapter just gathers these ideas together to aid in interpreting the demonstrations.

Section B5c presents two initial demonstrations of the system. After stepping slowly through a single task, it then demonstrates how the system's dependency network has grown through the experience. This demonstration, which we have already seen quickly in Section B3c, only proves the simple point that the dependencies accelerate the system in a situation that is precisely identical to one it has already been through.

Section B5d presents three more demonstrations. The demonstrations in Section B5c have proven the trivial point that dependencies accumulated during a given task can accelerate the system's performance on the same task. The more important question, though, is whether and when the dependencies will transfer to other tasks. In this the system achieves a mixed success that offers some important clues for later analyses.

Section B5e presents two final demonstrations. The first of these assigns the system a long task involving a compound goal. The system goes through some instructive gyrations to perform the task. Many issues arise along the way. The final demonstration repeats the same task, now with the benefit of the additional dependency network circuitry.

Section B5f interprets the results of these demonstrations in terms of larger points about the forms of perception and representation most appropriate to a situated agent. The chapter concludes with a historical conjecture about the ways in which the choice of blocks world as an experimental domain made these lessons difficult to discover and appreciate.

B5b Architecture

The running argument system has three motivations.

1. It is best to know what you're doing. Plan Execution is inflexible because actions are not based on an understanding of the current situation, only on the symbols in the Plan. The Planner once had a hypothetical understanding of why the prescribed action might turn out to be the right one, but that understanding is long gone. Flexible action in a world of contingency relies on an understanding of the current situation and its consequences for sensible action. (In case it isn't clear, I'm using phrase "know what you're doing" loosely and metaphorically. I do not mean to make any strong claims here about the nature of knowledge.)

2. One must continually redecide what to do. Decisions about action typically depend on a large number of implicit or explicit premises about both the world and oneself. Since any one of those premises might change, one must keep one's reasoning up to date. Each moment's actions should ideally be based on a fresh reasoning-through of the current situation.

3. All activity is mostly routine. Most everything you do during the day is something you have done before. This is not to say that one switches back and forth between two modes, one for routine situations and one for the occasional novel situation. Even when something novel is happening, the vast majority of what you are doing is routine. It has to be, otherwise you'd become hopelessly confused.

All this is more easily said than done. This chapter must explain how, as a technical matter, the running argument system instantiates these three ideals. In what way does the system know what it is doing? We think of "deciding what to do" as taking time, so how can the system be redoing it continually? How can the system determine which five percent of its reasoning needs to be conducted by non-routine mechanisms?

The running argument system engages in a rapid interaction with its simulated world. We can look at the system in two different ways, from the outside or the inside. First let's consider it from the outside. From the outside, the system is a fairly standard old-fashioned kind of rule system. We have already looked at this rule system in detail in Chapter B3. It has rules with left hand sides and right hand sides. Both left and right hand sides have variables. When the left hand side matches something in the database, the rule fires, the right hand side is instantiated with the appropriate variable bindings, and the resulting proposition is asserted in the database, perhaps causing other rules to fire in turn.

The system has a large set of rules, perhaps forty pages of code. Some of the rules concern the usual amenities of programming languages, such as boolean conditions in rules, simple forms of modularity, and macros that cause several assertions to be performed from a pattern. Some concern domain-independent ideas about actions, plans, evidence, issues, and arguments. Some concern stereotyped forms of domain-specific inference, such as the transitivity of 'above'. The most interesting rules concern domain-specific strategies and tactics and the arguments by which the system selects the applicable ones and chooses among them in particular situations.

The system behaves as if that entire set of rules ran completely to exhaustion, forward chaining until nothing is left to run, on every clock cycle. After the rules run, certain propositions in the database indicate which of its actions the agent intends to take on this cycle, perhaps picking up block B or moving right or standing still. In particular, the system has a proposition corresponding to each of its primitive motor capabilities. The motor system will decide what to do by checking which of these propositions are IN and which are OUT. The technical problem is that if we implemented this scheme literally, actually running all the rules on every cycle, it would be far too slow. The system ought to issue a new set of primitive motor commands about ten times a second. The rule system is fairly efficient, but it would still take it about five minutes,

running from scratch, to run all those rules. That is a real technical problem. One solution would be to apply brute force, carefully recoding the system for performance and running it on hardware that is three orders of magnitude faster. Such a solution would be an embarrassing kludge, offer no understanding of the problem, and scale very poorly. Fortunately, there is a better way.

Looked at from the inside, the system accelerates its operation by accumulating dependencies. Dependencies are helpful because life is mostly routine. Storing dependencies is a good investment because so much of what you do is something you are liable to do again. Furthermore, dependencies permit even a fairly general decision mechanism to operate in real time because so much of what you are doing at any given time is something you have done before. When a rule fires, the system builds a dependency record stating that a certain rule and a certain trigger lead to a certain conclusion. That rule need never fire on that trigger again. As the previous chapters have explained, we can think of a dependency network as a combinational logic circuit. Each proposition and rule corresponds to an electrical node in this circuit and each dependency record corresponds to a logic gate, an AND gate for IF rules and an AND-NOT gate for UNLESS rules.

This scheme offers technical accounts of the three slogans that formed our motivation. As the system runs, it accumulates this network. If the system gets into a routine way of life, goes the theory, then it should be able to run almost entirely out of the dependency network. The system knows what it is doing insofar as it effectively works through its reasoning from scratch rather than following a canned Plan. Combinational circuitry is highly parallel and extremely fast, so this reasoning takes little effort even if the circuit is a hundred deep. The system is continually redeciding what to do insofar as the circuitry produces a fresh set of decisions about action many times a second. Finally, due to the algorithms described in Chapter B3, the system can rapidly determine when its current situation calls for some novel rule-running. Even if a given cycle's decision is effectively based on hundreds of rules, the system only pays the price of the few truly novel rule firings. What is original about this system is not its 'compilation' of previous rule firings, but this graceful, automatic, incremental recourse to 'interpreted' rules when 'compiled' materials don't suffice.

Whether the running argument system actually does justice to our motivating slogans is an empirical question. Since no two situations in life are utterly identical, the system's success will turn on whether the dependency system really transfers the agent's lines of reasoning to appropriate future situations. If our current situation is ninety-five percent similar to something we have seen before, we ought to be reusing ninety-five percent of what we figured out before. As a detailed technical matter, then, does the system satisfy that ideal? That is what this chapter's demonstrations are about.

B5c Demonstration

Let's look at a simple demonstration. All the demonstrations take place in a simple blocks world. I chose blocks world because of its long history in the Planning literature (Sussman's Hacker (1975) was one of my influences at the time) and because its simulation and graphics code is easy to write. Figure B5.1(a)is a snapshot of the system before it has been asked to do anything. The horizontal line of bold dots is a table. The vertical line is just $x = 0$ and has no physical significance. The squares with letters A–B–C–D in them are blocks. The square with the stylized fingers is a hand.

The physics of this blocks world is very simple. On a given clock cycle, the agent can move the hand one unit horizontally and/or one unit vertically. Thus single-unit diagonal motions are allowed. If the bottom surface of the hand is touching the top surface of a block the agent can grasp the block. The hand has no state, so if the agent wishes to keep grasping the block it must continue asserting the grasping action. The world has gravity, so that unsupported blocks fall. It has no momentum, however, so that a moving hand can stop immediately when the agent stops commanding it to move. Blocks do not rotate. They obey velcro physics: one block will stay stacked on another so long as the bottom of the upper block is in contact with the top of the lower block, regardless of where the upper block's center of gravity is located. All of this will become clearer as the demonstrations proceed.

In this scene, the user has asked the hand to put block B on block C. Let's watch what happens cycle by cycle. I will explain what the numbers as we go along. In all cases, only the rough magnitude of the numbers is significant. (So don't worry if the numbers have been photocopied into illegibility.) The hand starts above and a little to the right of center of B. By the second frame, Figure B5.1(b), it has moved down and landed on B. This process has taken five cycles.

On the left side of the frame, some statistics are being kept to measure how much work the system is doing. Each line presents the statistics for one clock cycle. On the first cycle the number in the "rules" column is 229, meaning that the system ran 229 rules. This is the largest number of rules the system will run on any one cycle during these demonstrations, because this is the first time the system had ever been asked to do anything.

On the first cycle, it made two decisions: one to go left to get the hand centered on B and another to go down. As a result, it went left and down on that first tick. The reasoning that led to this step involved successively decomposing the goal into subgoals and planning ahead. Its first step was to get the hand on B; then it was going to have to grab B. All of the arguing this required took 229 rules.

Think of this rule language as a fairly general-purpose programming language. One could use it to write a Planner or a Scheme interpreter or anything else. This particular rule set implements a simple Planner using ordinary ideas about subgoal decomposition, preconditions, and the like. But it differs from most Planning systems in that it leads to a decision about what to do right now, effectively conducting the whole Planning

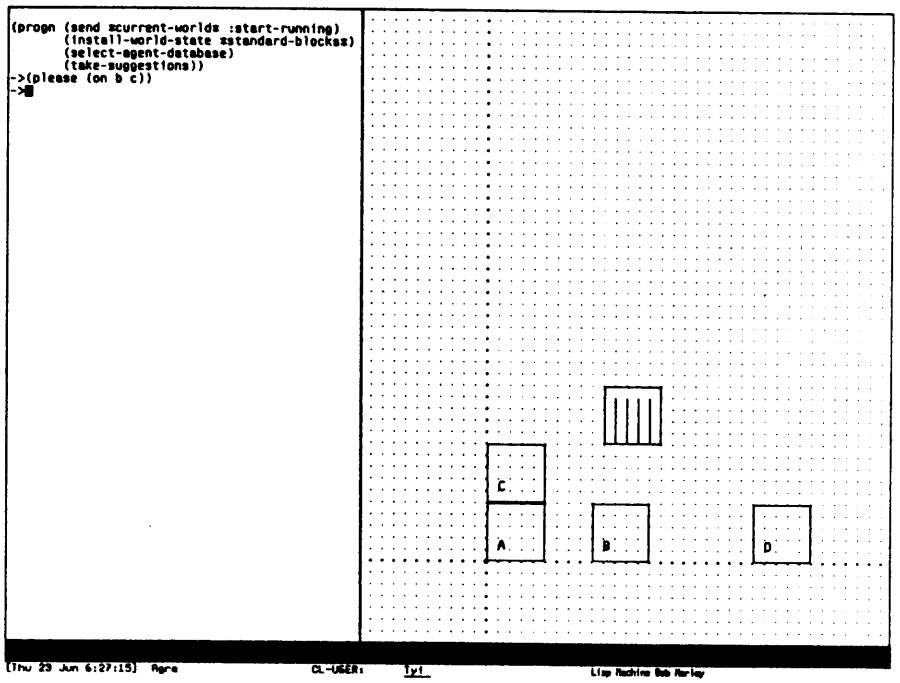

Figure B5.1(a). The user asks the system to put block B on block C.

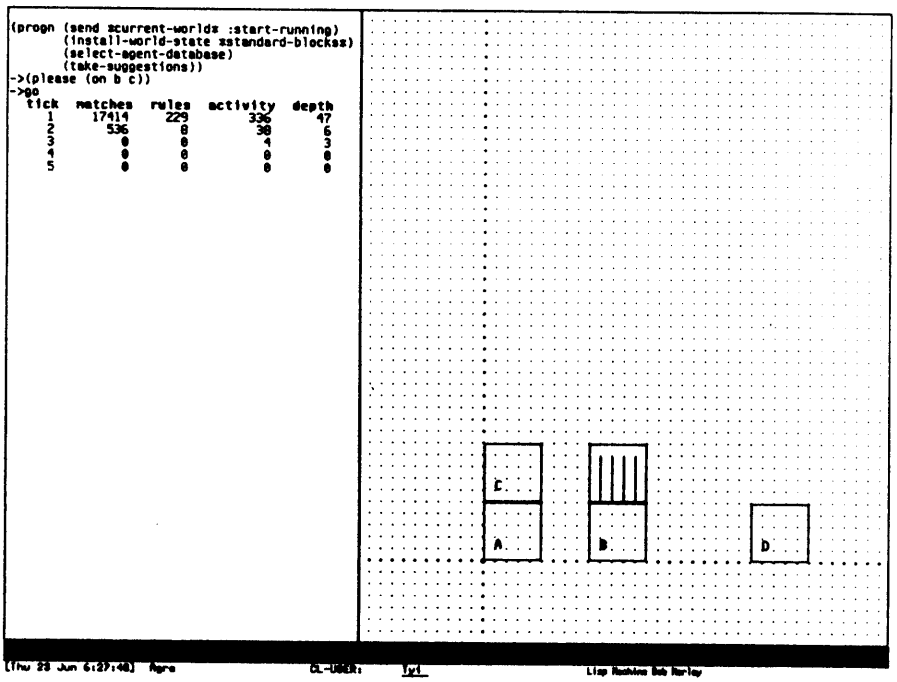

Figure B5.1(b). Once it gets under way, the system need not run any rules until the hand touches B.

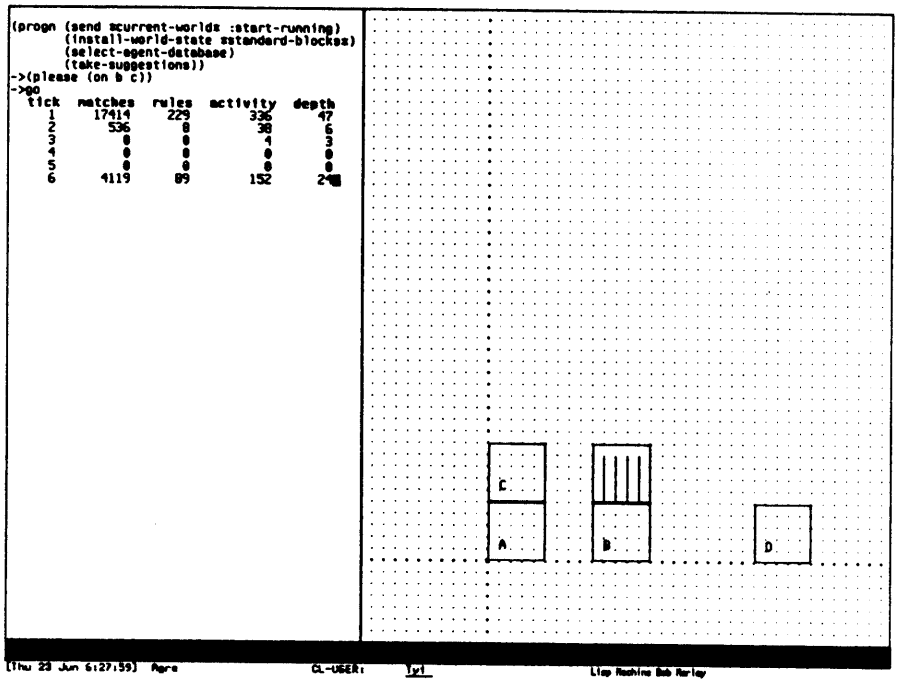

Figure B5.1(c). Never having picked up a block, the system has to run some rules to figure out how.

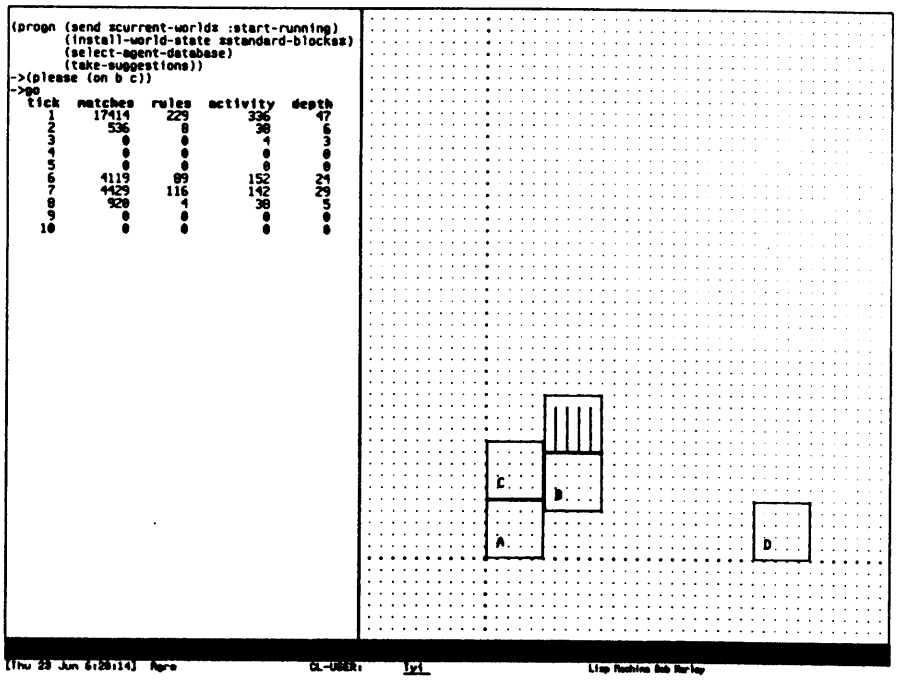

Figure B5.1(d). Never having moved a block anywhere, the system has to run some rules to figure out how, after which it runs smoothly until it strikes an obstacle.

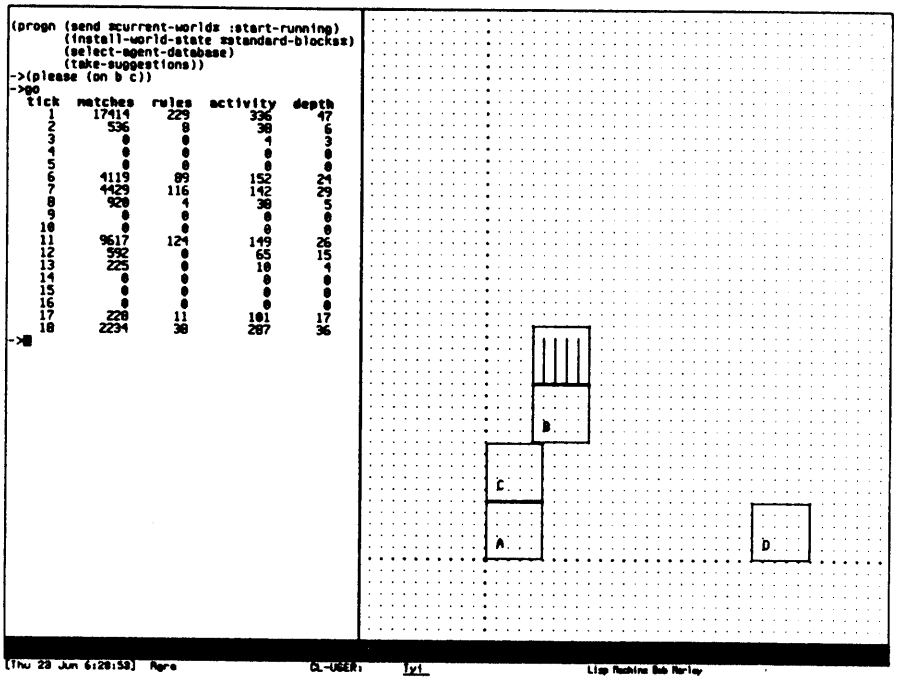

Figure B5.1(e). Never having hit an obstacle, the system has to run some rules to figure out how to circumvent it. Some more rules are required right at the end.

process afresh on every cycle.

The first cycle is atypical since the system is presented with the huge novelty of being presented with its first task. The subsequent cycles are more interesting. The hand only moves left for one cycle because it is only slightly off-center relative to B. But it goes down for five cycles. The decision to go down, according to the system's principles, ought to involve most of those 229 rule firings. Each cycle's incremental downward motion should result from a fresh reasoning-through of the situation, just in case something has come up to invalidate the previous cycle's reasoning. From the outside the system should appear as if it ran 229 all rules on all five of these cycles. But in fact it ran 8 rules in the second cycle and on the next three cycles it ran no rules at all. That is because nothing qualitatively changed in that time. The eight rules may have to do with the hand's ceasing to move left. Then it went down, step by step, and landed on B. Nothing qualitative changed, so no rules had to run. Figure B5.1(c) presents the situation just before the system has noticed that the hand is touching the block.

Figure B5.1(d) shows the scene exactly one cycle later. Sensory information has told the system that the hand is touching B, whereupon 89 rules run, a large number. But that is not all that happens on this cycle. Look at the number for cycle 6 in the "activity" column. On this cycle, 152 nodes in the dependency network changed their state. Much of that reflects the newly run rules; the system has never before found

its hand on something it is trying to pick up. But much of the activity also reflects the reasoning that is going out now that one of the reasons to keep going downward, mamely that the hand wasn't touching B, is now false. A wire that once read zero now reads one. That one has rippled through the network and now a lot of the arguments for moving down are unsupported. As they went out, zeroes went rippling through large parts of the network, thus accounting for some of the activity. The system didn't exactly decide to stop moving. The argument leading it to keep moving was no longer justified, so it is no longer moving.

On cycle 7 it takes the system 116 rules to decide to move B toward its destination. As before, the system must run a large number of rules since it has never moved a block toward its destination before. Once it gets moving, though, all the arguments that take place in those 116 rules continue to hold so the hand moves along without having to run any more rules.

Figure B5.1(e) shows the system after four more cycles. The hand has picked up B and has moved off toward C. The system doesn't have a lot of foresight and doesn't know very much about trajectories. It has chosen its motions serviceably, but nonetheless it has bumped into C. What happens now, on cycle 11? There had been an argument for going up, which was that B was below C and needs to be above C and the hand is holding B, so the system moves its hand up. There was also an argument for moving left, which is that B needs to be overlapping horizontally with C, and B is all the way to the right of C, so it is moving left. This argument for moving left is still good since B is still to the right of C. But now the side of B is touching C. That fact is causing some rules to fire that had never been involved before. The rules object to moving left on the ground that it is not a good idea to push the object you are moving toward. This objection, like all objections, is only offered *ceteris paribus.* But since no rule objects to it in turn it is accepted and the proposal to move left is defeated. All this action takes 124 rules. The argument for going up is still wholly uncontroversial, though, so the hand continues moving upward.

When it gets up far enough that B clears C, the argument against moving left no longer holds, so there is a lot of activity in the network as that objection goes out. Henceforth it is free to move left. The proposal of moving left has been active all the while, but it has not been adopted because of the objection overriding it.

On the final cycle, B is now on top of C. This is velcro physics, so slightly on is on. It finishes, running 38 rules because this is the first time it has ever finished a job. Those rules are associated with detecting that the job has been finished and perhaps with cleaning up. The network is extremely active on this final cycle, with 287 nodes changing state, a large portion of the whole network. Those 287 nodes represent the whole apparatus of inference and argument behind the top-level goal, its decomposition into subgoals, and the idea of having the hand on something and moving it along. Now that B is on C, the goal has been achieved, so the support for all of that apparatus has gone out. Where once was a one is now a zero, so a bunch of zeroes propagate through the network, and the network settles down to its rest state. Large sections of

the network that turned on during the first cycle now turn off during the last cycle.

Some patterns in these numbers indicate some of the important dynamics of the system. Most of the patterns concern the levels of activity in the network. Observe that on the first three cycles an initial burst trails off to zero: 336 to 38 to 4 to 0. Then a second burst starts on cycle 6. That burst lasts for two cycles, one for grabbing B and one for getting moving, but then like the initial burst it fades to zero. Then starting on cycle 11 is a third burst as B hits C and beings sliding up along it. At the end are two bursts on adjacent cycles, for clearing C and then for finishing the job. This burst-decay pattern will be ubiquitous during all the demonstrations.

To understand this burst-decay pattern, think of the dependency network as moving through a gigantic phase space with a dimension for each node. As the system works on its task, it describes a trajectory through this space. When nothing is qualitatively changing, the network has no activity. When the network crosses a boundary into a qualitatively different region, such as when it encounters an unexpected situation or moves from one subgoal to another, then the network will change its configuration to reflect the change. Patches of the network that had been active will turn off and other patches that had been quiet will become active in their place. Each burst of activity reflects this sort of change. The burst usually doesn't decay to zero immediately because of proprioception; the system receives a signal from its body indicating that an attempt to move the hand or to grab something is actually succeeding. Observe that the burst-decay pattern only concerns the amount of activity in the network, not the number of rules that fire. In later demonstrations will see the "activity" number repeatedly burst and decay without any rules at all being fired.

A second pattern concerns the relative magnitudes of the activity numbers. The network is most active at the very beginning and the very end. At the beginning it is starting with a top-level goal and it is decomposing that progressively down into subgoals. That very top-level goal and its associated structure remain IN during the whole process. A region of network configures itself for putting B on C and it stays in that configuration until the end. The numbers in-between tend to be smaller because they reflect smaller events. The other peaks of activity occur when the system switches from one major subgoal to the next. Another peak occurs when the system must pull up short and reason about an unexpected condition, B's hitting C; and then again to retract all that reasoning when the unexpected condition goes away.

This pattern becomes even clearer if we look at the "depth" numbers, which reflect the length of the longest chain of nodes in the network that changed state—that is, the longest causal chain within the network on this cycle. This number will tend to relate to the depth of the reasoning patterns. The depth is very high when the system is decomposing its top-level goal on the first cycle and when it retracts all the reasoning behind the decomposition on the last cycle. In more complex examples we'll see that pattern nested within itself.

The system has now finished performing its first task. In doing so, it has had to run several hundred rules, but it has also built up several hundred gates worth of dependency

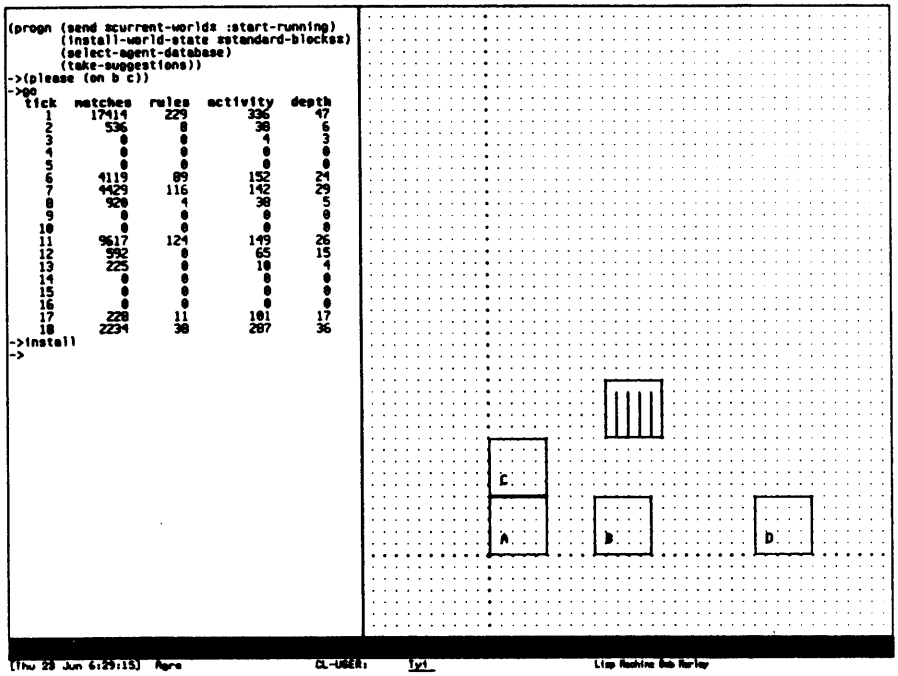

Figure B5.2(a). Here is the first demonstration run over again.

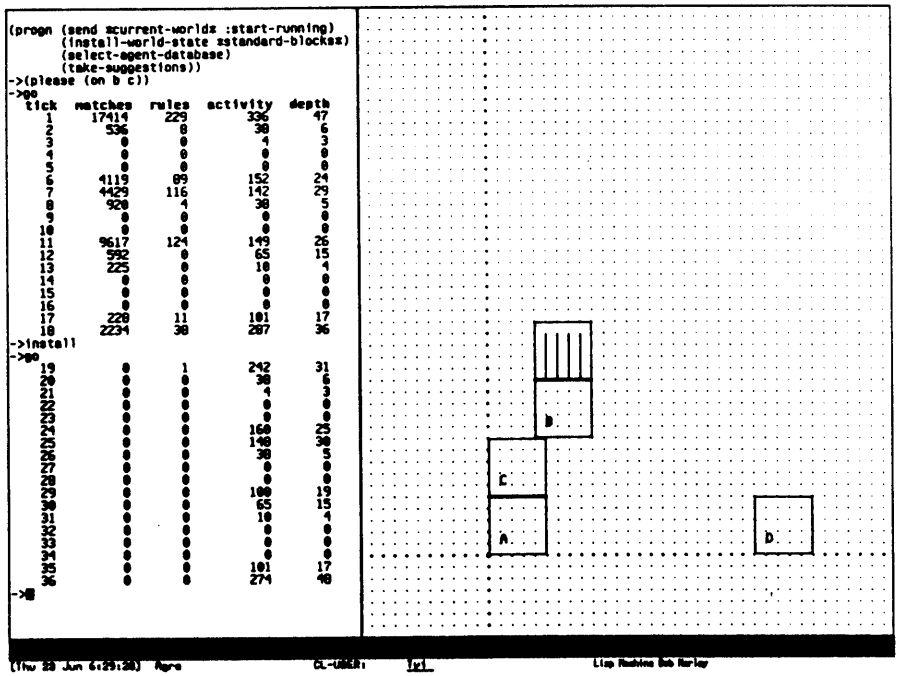

Figure B5.2(b). The user has restored the blocks to their original positions and set the system running again with the same goal. After running a single rule to recover from the discontinuity, the system can run entirely out of its dependency network.

network. If the system is working well, those dependencies ought to permit the system to run fewer rules in the future. Later sections will present some detailed experiments to assess whether the system's experience putting B on C transfers in satisfactory ways to other activities. For the moment, though, let's just test the simplest case. Do the dependencies permit the system to perform exactly the same task in exactly the same situation without running a significant number of rules?

Figure B5.2(a) shows the system after the blocks and the hand have been restored to their original positions. (This will slightly confuse the system for a moment because the discontinuous change will foul up the computation of the proprioceptive information.) The same request, putting B on C, is still in effect. Then we set the system running. It runs a single rule on the first cycle (to recover from the discontinuous change), but after that it runs absolutely no rules at all. It makes exactly the same moves it made before, including bumping into the side of C. The system is not learning in the sense of adaptively changing its behavior; it is just doing the same dumb thing much more efficiently. The system is firing no rules and performing no pattern matches, but the numbers for the activity in the network are qualitatively the same. The activity is very high at the beginning and end and exhibit peaks at significant transitions. We can observe the same repeated burst-decay pattern in the activity levels. Likewise, the depths are the largest at the beginning and end, with intermediate values when the system changes subgoals or when something exceptional happens. From the outside, the system appears running all the hundreds of rules it ran before. From the inside, though, no rules are firing because the dependency network covers every case that comes up. An observer sitting at the Lisp Machine console sees the system move along smoothly and rapidly, whereas before the system hesitated at important transitions.

As I mentioned, this test is not very difficult. Of course the dependencies suffice to perform precisely the same task again. This result is entirely independent of what the rules say. The subsequent demonstrations rest more heavily on the ideas about arguments presented in Section B4d.

B5d Patterns of transfer

To evaluate how well the running argument system lives up to the slogans that form its motivation, let us consider five more demonstrations of the system in action. We are concerned with whether and when dependency maintenance accelerates the system, so each demonstration except the last picks up where the original demonstration left off, with the system just having been asked to put block B on block C. In having performed this task, the system has built several hundred gates worth of dependency network. The question now is, what can the system now do automatically, running out of its dependencies instead of rules, in virtue of having had that experience? We saw that you can do precisely the same thing; that is not very interesting. The question is, what can you do that is *not* precisely the same? What makes a situation sufficiently similar that the effort of building that hunk of dependency network is going to transfer? How

much transfer does the system exhibit? Do the system exhibit all the transfer we feel it ought to, given the slogan that most everything you do is something you have done before?

The third demonstration is a success story. In Figure B5.3(a) we have backed up to just after the completion of the first demonstration and restored the four blocks and the hand to their original positions. Now, instead of asking it to put B on C, we are asking it to put B on D. It is a different task, but both tasks involve picking up B.

On the first cycle, number 19, the system runs 34 rules. That is not bad given the 229 rules it ran on the first cycle during its first task; many of those rules must have carried over to this second task. Those 34 rules presumably concern the different destination.

On cycle 24 the hand stops upon reaching B and decides that it should grab B. The first time out, during the task of putting B on D, that took 89 rules. This time it takes one rule. The activity number is still high, 152 as compared to 160 before, reflecting the network's change in configuration as one region of the network goes OUT and another comes IN.

On the next cycle, number 25, the system decides to pick up B and move it to the right towards D. That takes 85 rules, as compared to the 116 rules it took to decide to move B to the left toward C during the first demonstration. Evidently little has transferred from the first task to the second, but it is hard to say how much ought to have transferred. Certainly moving left toward C and right toward D are different activities.

The system completes its task without further incident on cycle 35. The final cycle, on which it realizes it is done, takes 29 rules, as compared to the 38 rules it took to complete the first task. The activity number is high and approximately the same in both cases, 262 as compared to 287 before, reflecting all of the apparatus that has gone OUT once the system has achieved its goal. Again, little appears to have transferred from the completion of the first task to the completion of the second, but it is again hard to say how much ought to have transferred.

Despite the uncertainties, this demonstration offers us two clear instances of dependencies constructed on the first task carrying over to the second. On the very first cycle, much of the apparatus of decomposing the goal and initiating action appears to have been independent of the particulars of the two tasks. And when the hand found itself on B and decided to grab it, the apparatus associated with that subgoal was independent of the larger goals of which it was a part.

The fourth demonstration also begins after the system has completed its first task of putting B on C. The initial situation is precisely the same except that all have different names now. Instead of A–B–C–D we have E–F–G–H. The task now is putting F on G. Thus the situation and goal are both exactly isomorphic to the originals except for the names of the blocks. Are the situation and task the 'same' as before? Yes and no, but we'd really like the dependency structure from the first task to carry over here. After all, the system's actions and the reasoning behind it will have exactly the same form as

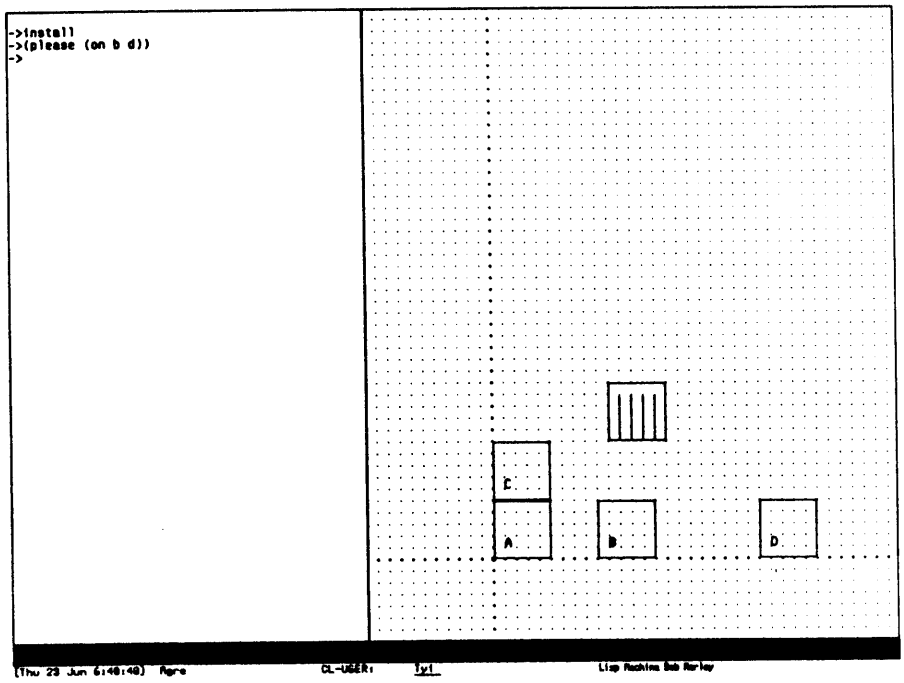

Figure B5.3(a). The blocks have been restored to their original positions and the system has been assigned a slightly different task, putting B on D.

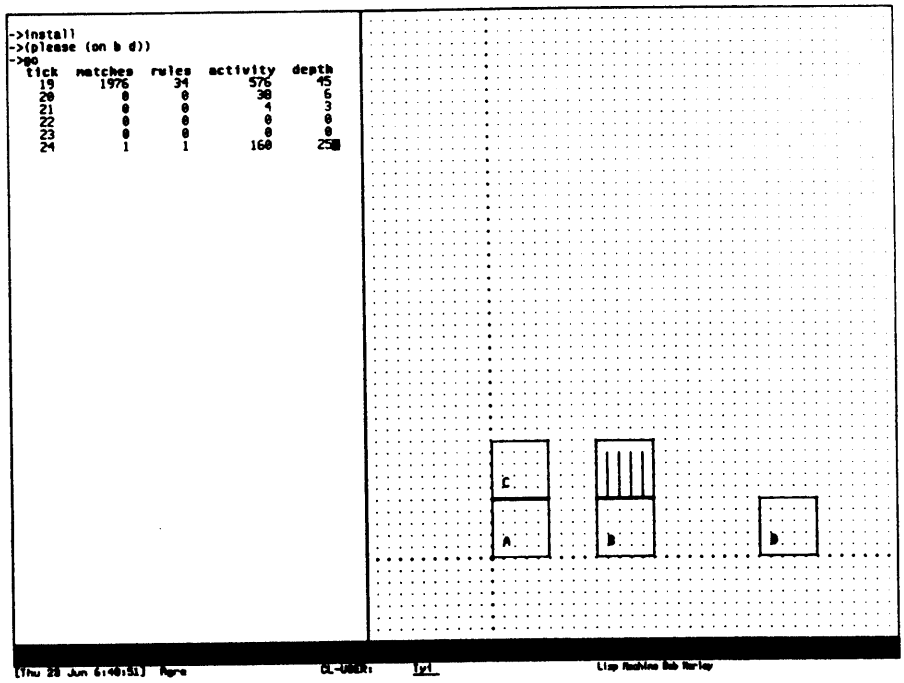

Figure B5.3(b). The system has already picked up B in the previous task, so that portion of the dependency network transfers to the new task.

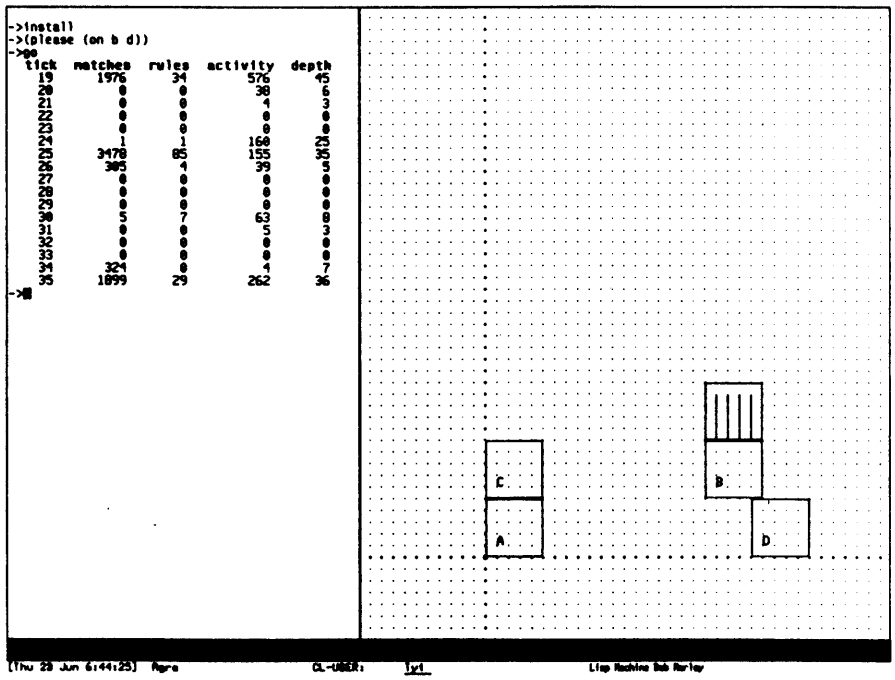

Figure B5.3(c). The rest of the task requires somewhat fewer rules than before, but the dependencies do not transfer to putting something on D.

before. How much *does* transfer? Not enough.

The numbers prove it. The numbers of rule firings at major transitions range from half to three quarters of the corresponding numbers over the first demonstration. It takes 159 rules to get moving. When the hand lands on F it takes 45 rules to grab it and 64 rules to get it moving. It takes 96 rules when it hits the obstacle. Notice that the decay portion of the burst-decay pattern has transferred; all the system's reactions to proprioceptive information have transferred. Much that was independent of the particular individuals has transferred. For example, the system has already unfolded some relatively domain-independent apparatus having to do with arguments and plans. But nothing that relates to these particular blocks has carried over. If the system runs a rule like

```
(if (and (trying (grasp ?x))
         (on hand ?x))
  (propose (grasp)))
```

then this rule will have to run again for every block that the system is ever trying to grasp. During the first demonstration it ran with x bound to B. During the second and third demonstrations it did not have to run because the system did not ever try to grasp any block except B. During this fourth demonstration, though, it tried to grab F. As far as the system's representation scheme is concerned, F is a wholly different block from

B, so all the rules that originally mentioned B must now be fired again with `x` bound to F, thus creating a duplicate, parallel dependency network structure. None of this makes the dependency network any deeper, but it does make it bulkier. This proliferation of network stuff doesn't have any principled end. So long as new blocks keep coming, the system will keep having to build more network stuff.

Even worse is a rule that mentions two blocks. Consider the following rule, simplified in various ways for purposes of exposition.

```
(if (and (propose (move hand ?direction))
         (horizontal ?direction)
         (grasping ?x)
         (sides-touch ?x ?y)
         (in-direction ?direction ?y ?x))
  (propose (object (move hand ?direction) (dont-push ?pushed))))
```

This rule, or actually a more general version of it, posted an objection in the first demonstration when B accidentally bumped into C. It also posted an objection on the fourth demonstration when F accidentally bumped into G. Each time, the system had to build a new patch of network structure. The system is faced with the possibility of having to build an amount of dependency stuff proportional to the square of the number of blocks it encounters. This might be tractable in some domains, but it certainly isn't very satisfying.

The running argument system might be at its worst on an assembly line. Asked to perform analogous tasks in an endless stream of analogous situations comprising different individuals, the system will generate a vast number of analogous circuits, none of which it will ever use again. Imagine passing a thousand cars on a long car trip, turning the knobs on a thousand doors over the course of a year, turning a thousand pages while reading comic books on the beach, or eating a thousand spoonfuls of cereal over successive breakfasts. Each series involves interactions, each perhaps subtly different, with a thousand different individuals whose individuality *per se* has little effect on what actions are correct. The knowledge embodied in the running argument system's rules is abstracted away from the particulars of a thousand situations with the use of variables. But dependencies do not support variables, nor can dependencies be generalized to support variables without losing most of the virtues of dependencies.

The lesson of the two foregoing examples is that reasoning involving the same individuals will transfer, even into different contexts. But no transfer will take place between one set of individuals and another. The system decomposes its reasoning automatically but it doesn't make analogies automatically.

The fifth demonstration proves that the rule set isn't perfect. Here we are picking up from the end of the first demonstration without restoring the blocks or hand to their original positions. The new task is to put C on B. It would be easy enough to write rules to exercise some foresight in such cases, but I haven't. The hand shuffles to the right around B and then pushes it to the left on its way to grabbing C. It now reasons that the hand is on C, that C ought to be on B, that B is higher than C, and that therefore

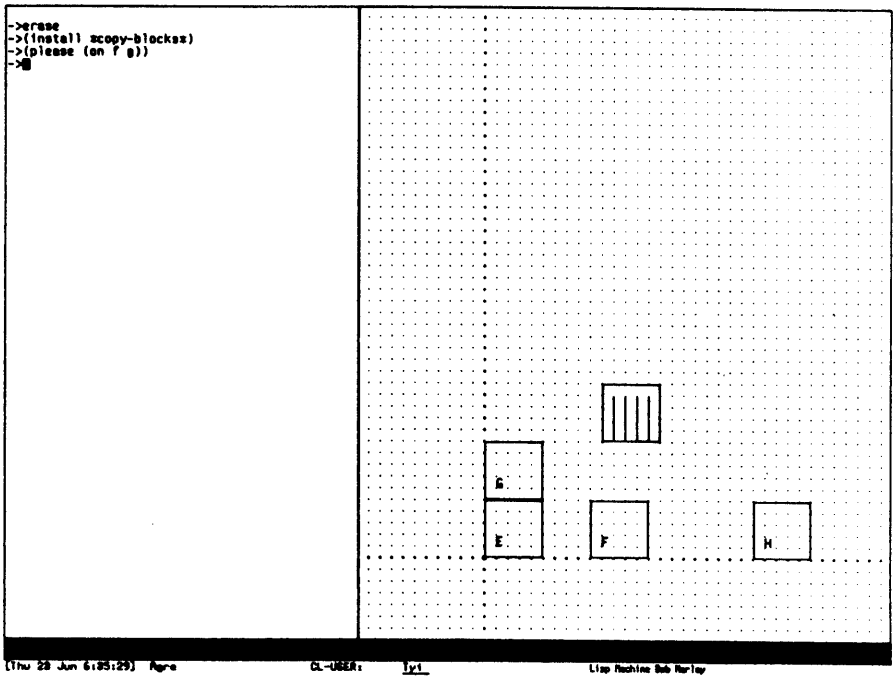

Figure B5.4(a). Here the original blocks have been removed and replaced with identical blocks that occupy the identical locations but have different names.

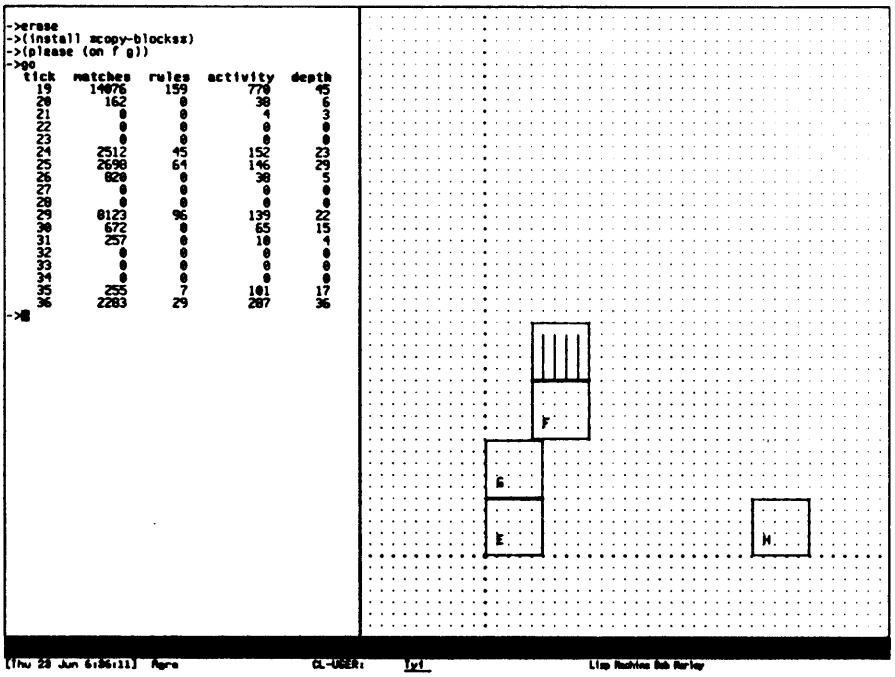

Figure B5.4(b). Having been set to a task that is perfectly analogous to one it has performed before, the system must run many rules.

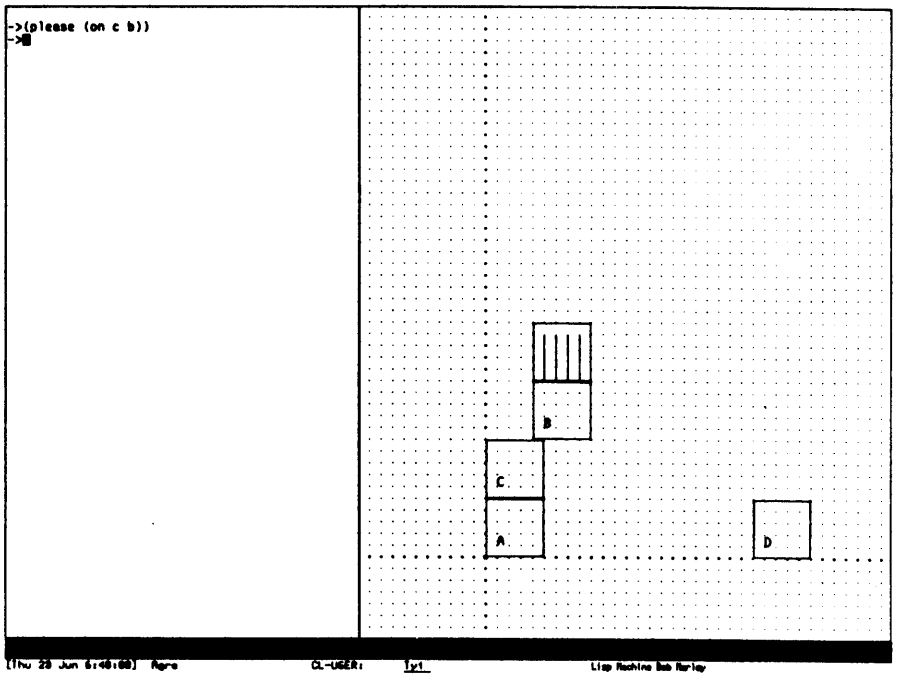

Figure B5.5(a). Picking up from the end of the first demonstration, the system is asked to put C on B.

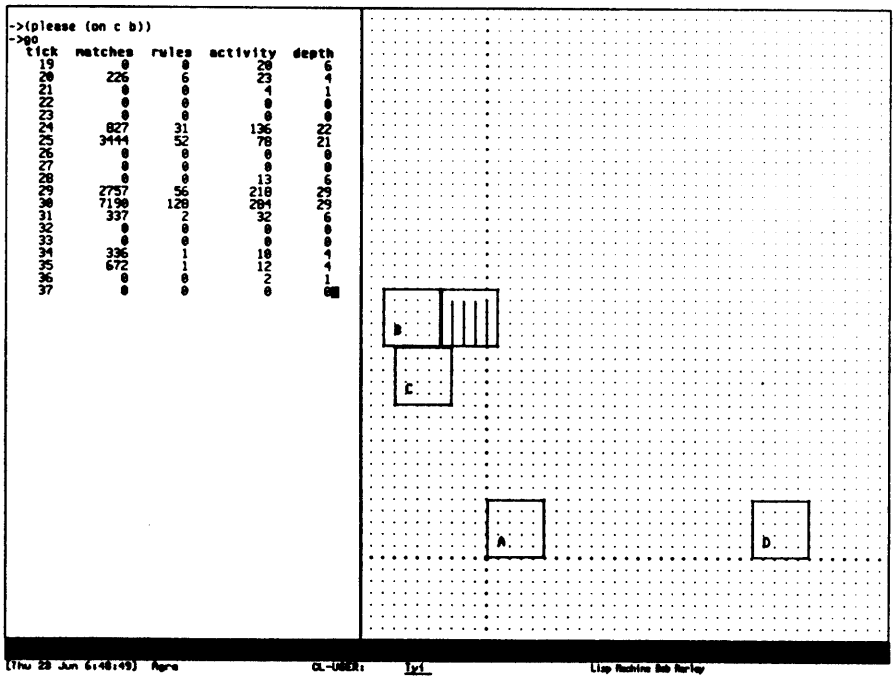

Figure B5.5(b). Unfortunately I hadn't written rules to cover this case, so it messes up.

it should move the hand upward. A separate line of reasoning leads it to move the hand leftward as well since it would like to center C on B. Both these lines of reasoning proceed to apply on each next cycle *ad infinitum* as the hand and blocks float upward and leftward off the screen. The dependencies capture the bogus lines of reasoning right away, so once the hand and blocks get going they move along with no hesitation. They have the nonchalant quality of a balloon taking off. Naturally something more realistic and interesting would have happened had I paid more careful attention to modeling the practicalities of arms, hands, and motor control.

B5e Transfer and goal structure

The sixth demonstration is longer and, having been chosen for its twists and turns, illustrates some additional dynamic effects. Once again the system has just put B on C and the blocks and hand have been restored to their original positions. The task now is a sequence of two subtasks, putting D on C and then putting A on B.

The "and" in the task description is not a logical conjunction but rather an instruction to perform the subtasks in sequence. Originally "and" was a conjunction but it got to be too much trouble to implement the conjunctive semantics. The reason was quite annoying. Naturally when the system faced two tasks it needed rules to determine which one to perform first. As with all its decisions, the system needed to conduct an argument with itself, putting forward proposals and weighing the arguments for and against them. In many situations it is plain which to do first. Perhaps the hand is already perched on the block it would have to move first to perform one of the subtasks. Or perhaps the blocks need to be stacked in a particular order. Rules for cases like these were easy and fun to write. The rules were harder to write when there was no reason at all to choose one task rather than another. Even though it didn't matter which it chose, the rule language had no way of expressing an arbitrary choice. All the rules fired on all the individuals to which they applied. I considered extending the rule language with a mechanism for performing arbitrary choices, but I couldn't come up with anything sufficiently principled and general. I ended up writing rules that implemented utterly arbitrary decision schemes, but no one criterion sufficed to discriminate in all cases, so the criteria then had to conduct an elaborate argument among themselves. This did work, but nothing was gained in waiting for all the hundreds of necessary rules to fire. The issue is deeper than it looks; I will return to it after the demonstrations.

On the first cycle, number 19, the system has a lot of work to do. It has never been asked to perform a sequence of tasks, so it must run many rules to decompose the compound task, assign itself the subtask of putting D on C, and figure out how to head toward D. All of this action takes 251 rules, quite a large number. Once it gets moving, it puts D on C without incident. The burst-decay pattern occurs repeatedly throughout the process. Deciding to grab D takes 46 rule firings and deciding to move D upward and leftward toward C takes 72 rules. Both of these numbers are probably excessive, given that the system has already put B on C, a fairly analogous task.

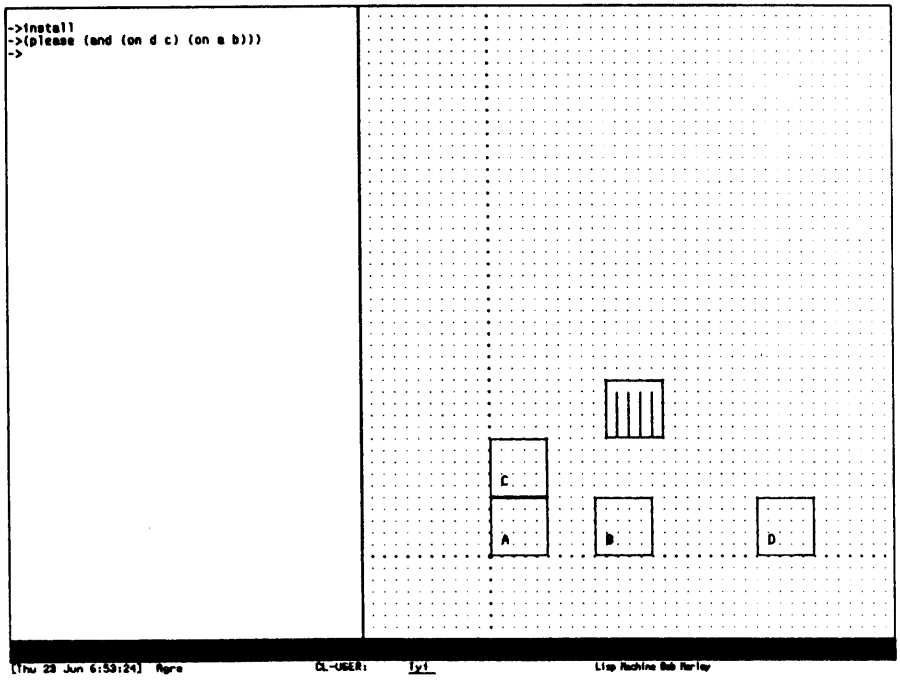

Figure B5.6(a). The blocks have been restored to their original arrangement and the system has been assigned a complex task.

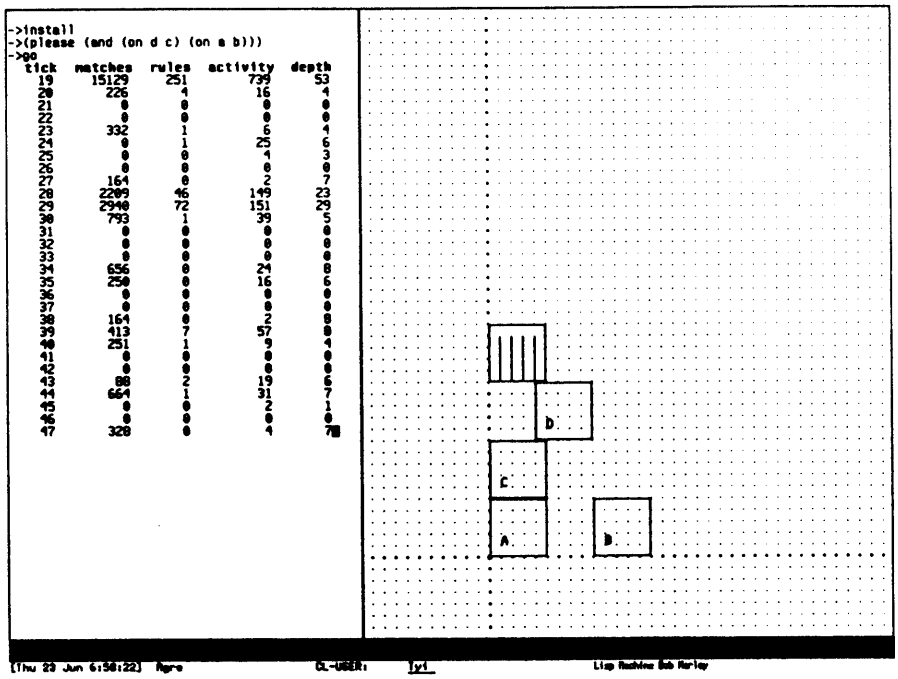

Figure B5.6(b). It places D on C without incident.

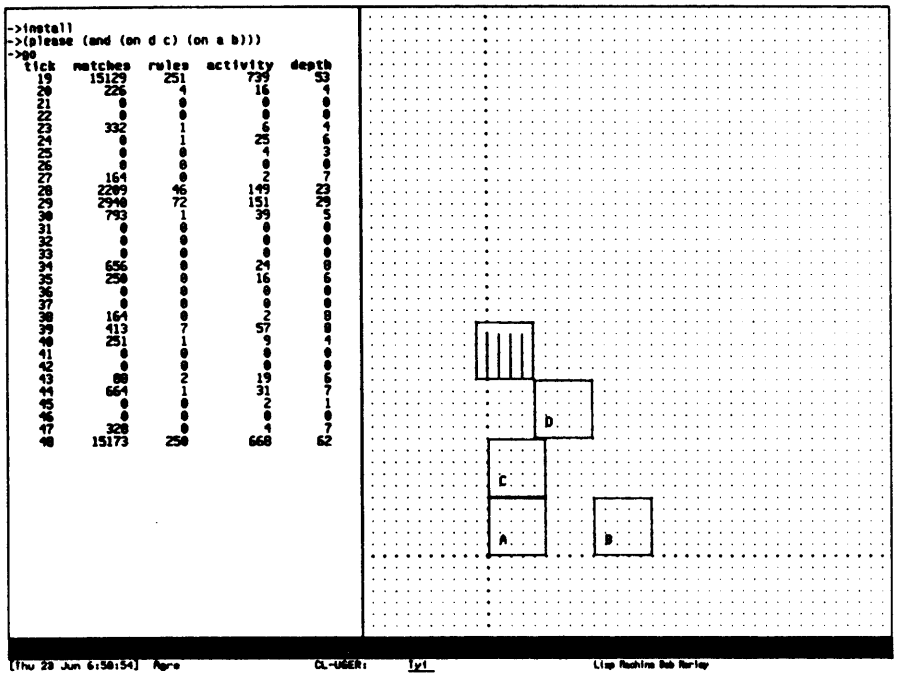

Figure B5.6(c). It prepares to move the hand around D on its way to pick up A.

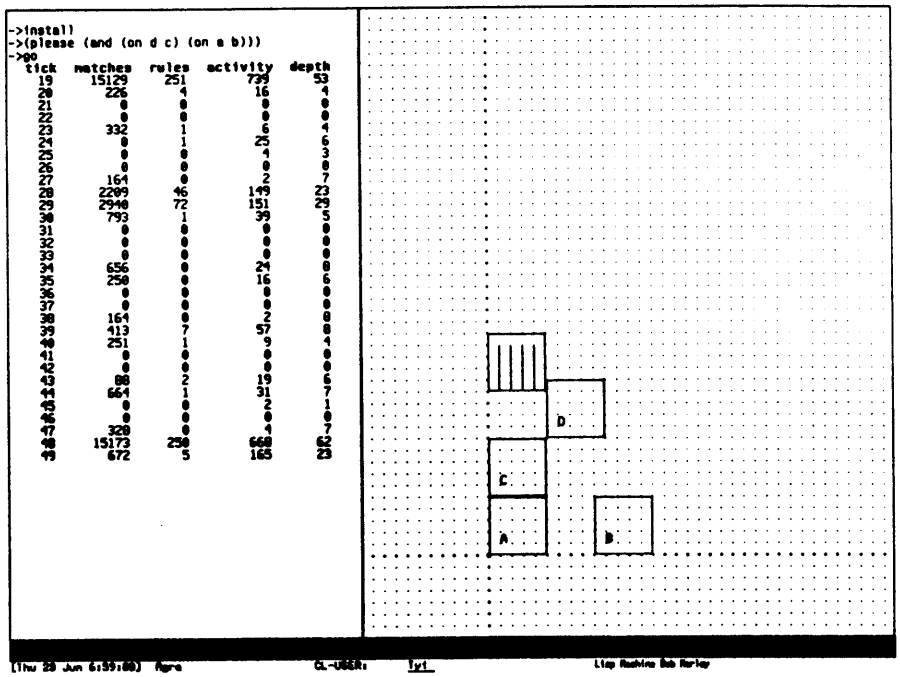

Figure B5.6(d). Centering the hand over A causes it to bump D off of C, thus undoing the first subgoal.

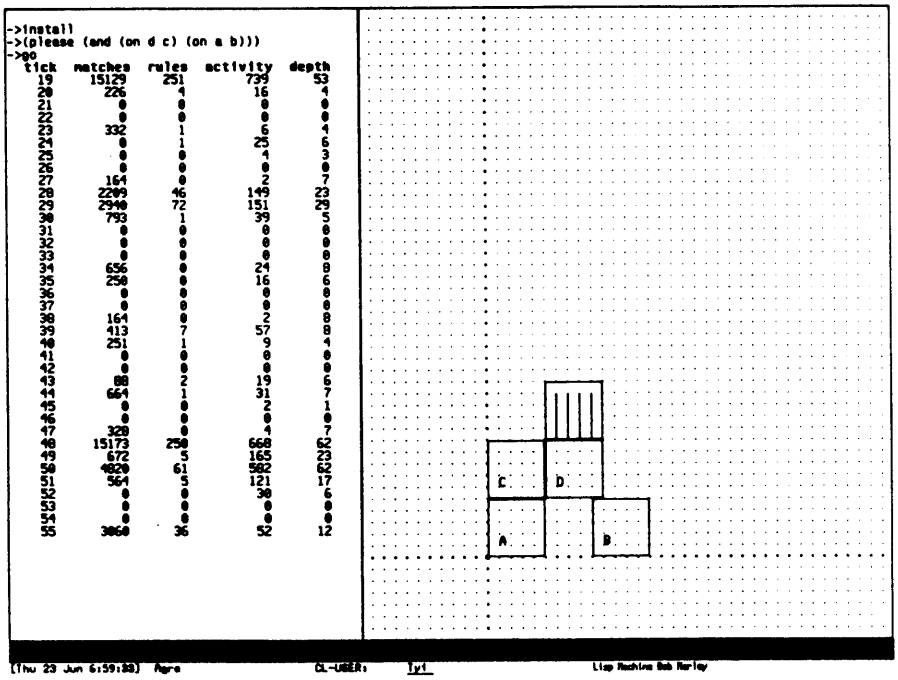

Figure B5.6(e). Shifting back to its pursuit of the first subgoal, it chases D as it falls.

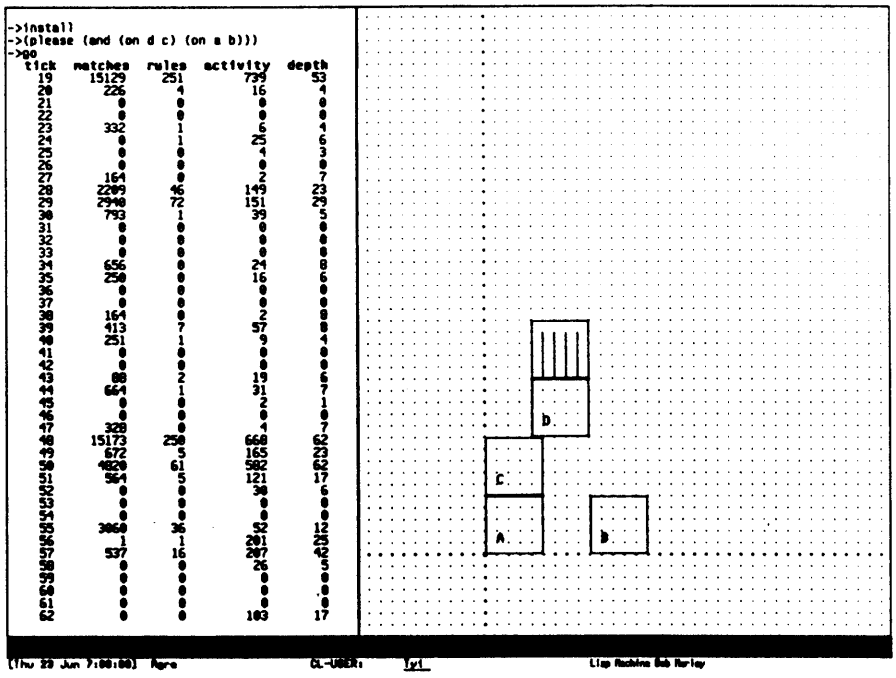

Figure B5.6(f). Having caught D again, it puts it back on C.

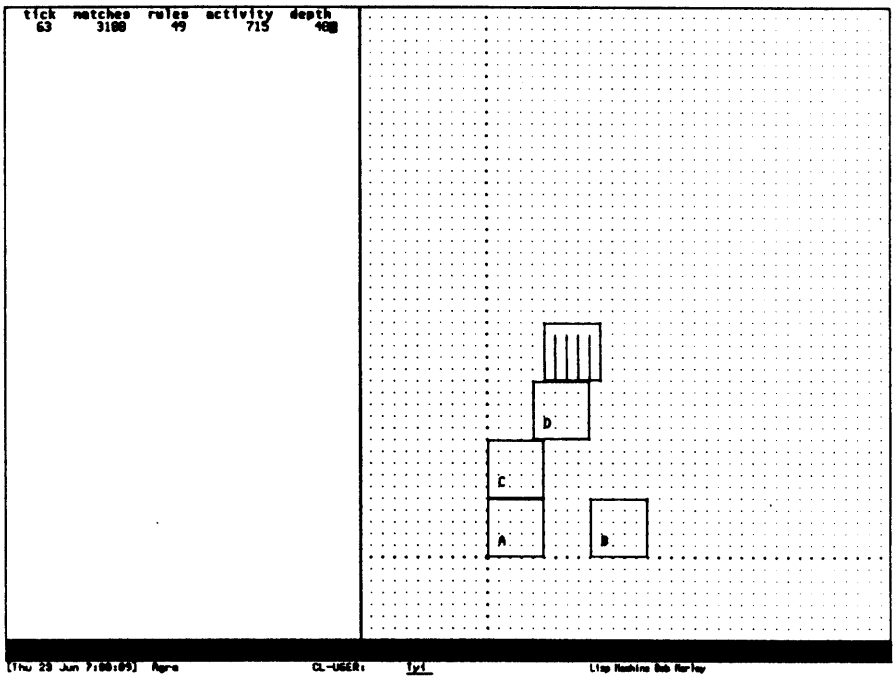

Figure B5.6(g). This time it decides to go around to the right.

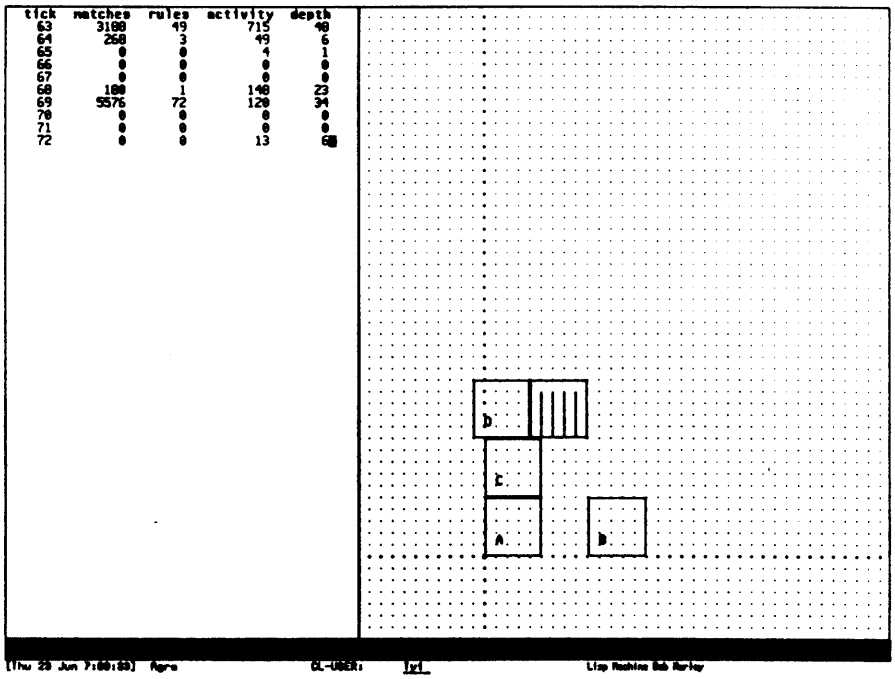

Figure B5.6(h). Heading for A, the hand strikes C.

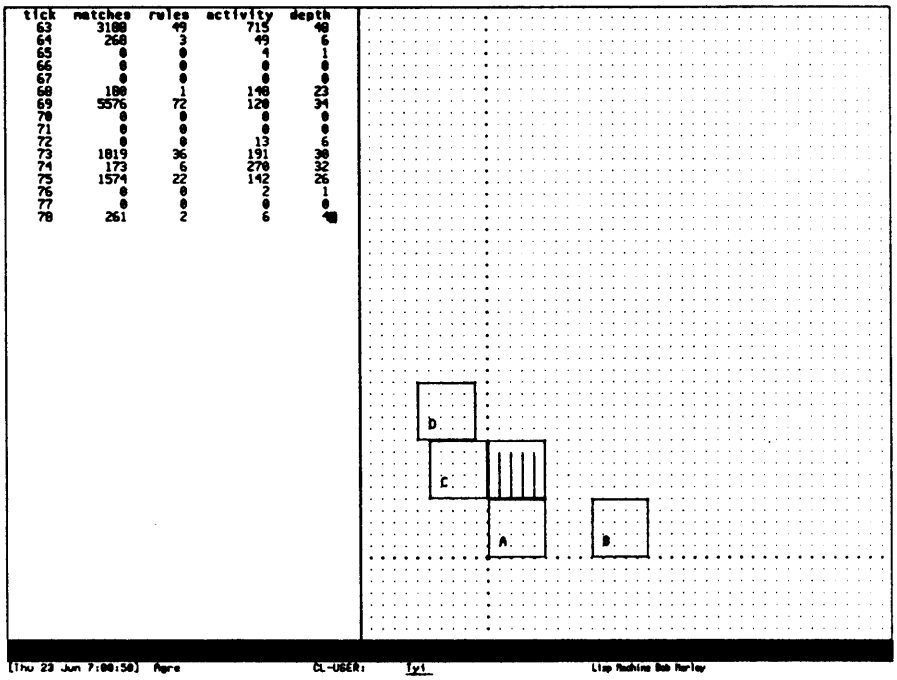

Figure B5.6(i). Working its way around C, the hand arrives on A and grabs it, pushing C and D off along the way.

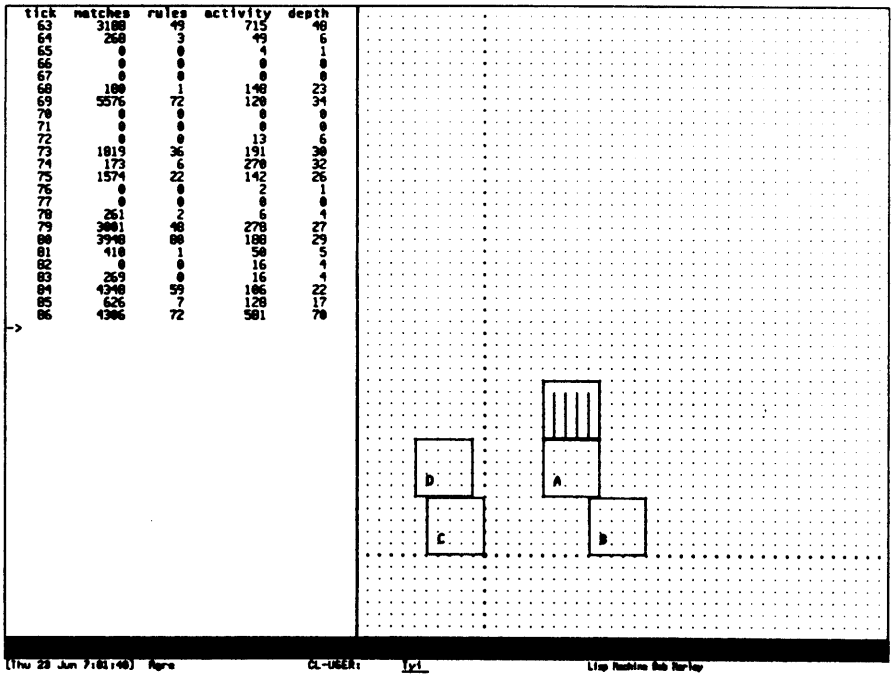

Figure B5.6(j). Finally it places A on B without incident.

Figure B5.6(b) shows the system after cycle 47, immediately after the system has put D on C and is about to discover that it has finished with the first subgoal and then get started on the second. A great deal happens in the next few cycles.

Something complicated happens on cycle 48. Having finished with its first subtask, the system is pursuing the task of putting A on B. Its first subsubtask is to get its hand on A. A is below the hand so the system proposes moving downward. But then the objection arises that the hand cannot move down because a stack of blocks is directly below it. (The system anticipates the problem without trying to move the hand downward.) Another rule then offers the alternative of going around and proposes moving left. There being no objections, the hand moves left. This took 250 rules, indicating quite a complex process of a sort the system has never before performed. The level of activity in the network is very high, 799, reflecting both this reasoning coming IN and the reasoning behind the first subtask going OUT.

Something unfortunate happens on cycle 49. The proposal of moving down to get on top of A, having been made on the previous cycle, still applies. And the objection against it, that the stack of blocks is in the way, no longer applies, so the hand moves down. Another rule points out the virtue, other things being equal, of centering the hand over the block, and thus proposes moving to the right. No rule objects to the unfortunate combination of proposals, so the system adopts both and the hand moves downward and to the right, pushing D to the right off of C. (This is not the sort of mistake that most blocks-world Planners would ever make, but then then it couldn't even happen in most blocks worlds, which are even less realistic than mine. For an exception see (Fahlman 1974).) All this took only 5 rules since almost all of this reasoning had taken place in other contexts. It did involve an activity of 165 in the dependency network, much of which was no doubt due to objections from the previous cycle that no longer applied and so went OUT .

Cycle 50 is the most interesting. Since D is no longer on C, the first subgoal does not hold true, so one of the justifications for pursuing the second subtask has been removed. The activity is very high, 582, as the network changes configuration, sending the second subtask's apparatus out and bringing the first subtask's apparatus back in. The process takes 61 rules, many of which reflect D's top surface being above the bottom surface of the hand, leading the system to conclude that it must move the hand upward. Some of them also reflect the hand being in side-to-side contact with D, leading the system to defeat the proposal of moving to the right.

Starting on cycle 51, D falls and the hand begins chasing it rightward and downward. The hand's transition from upward to downward motion occurs mostly courtesy of the network, having transferred from the first time it fetched D. Finally the hand lands on D on cycle 55. On cycle 56 it decides to grasp D, which reasoning carries over entirely from the first time it grasped D, except for one stray rule. On cycle 57 the hand lifts D straight up, having run some rules to defeat the proposal of moving leftward as well. When B bumped into C during the first demonstration, it took 124 rules, much of which has transferred over to this case despite the different individuals. Much of this transfer

does not concern bumping-into *per se*; cycle 11 was the first argument of any difficulty that the system had conducted with itself on any topic. From cycles 58 to 62 the system moves D back up onto C with no rule firings at all.

On cycle 63 the activity number is very high again, 715, reflecting another grand swap as the first subtask's apparatus goes back OUT and the second subtask's apparatus comes back IN. The system does run 49 rules on rule 63; these concern the old problem of getting the hand around the stack of blocks and onto A. This time, the hand is right of center instead of left, so some rule proposes moving to the right around the blocks. The system adopts this proposal and moves right. As before, objections defeat a proposal to move downward.

The rest of the trip is comparatively uneventful. The hand clears the right edge of D on cycle 67. On cycle 68 it begins moving downward and to the left toward A, inadvertently pushing D to the left as it goes. Fortunately it does not push D off of C again. Instead it lands on C on cycle 72 and repeats the same trick, moving to the right one step and then changing course again to move downward and to the left. This motion pushes D to the left as well.

The hand moves to the left all the while in order to get itself centered on A. Several rules offer objections and counter-objections around the issue of whether it is OK to push the blocks out of the way; in this case they approve. Although the line of reasoning for circumventing C was closely analogous to that for D, we observe little transfer because D and C are different individuals. Some general-purpose reasoning about getting around things did transfer, but not very much.

Finally the hand lands on A on cycle 78, just in time to push D and C completely off of A. Some rules fire on cycles 79 and 80 because the system has never picked up A or moved a block to the right before. (Left and right, like the block names, are individuals across which transfer fails as well.) Some additional rules navigate A over top of B when it accidentally bumps into it; again we do not see much transfer from previous such cases. Finally the system completes its task on cycle 86.

Before looking at the numbers in more detail, let's go through a seventh and final demonstration. This demonstration, unlike the others, picks up from the sixth. The system has just finished with the compound task of putting D on C and A on B. We have restored the blocks to their original positions and set it running again. The same compound goal is in effect, so the system marches through precisely the same actions as before. It takes a couple rules to get moving. The activity numbers exhibit the usual burst-decay pattern. The system has not gotten any smarter about performing this task but it has gotten faster.

In both the second and seventh demonstrations, the system was running through a task it had already performed. In each case, the system went through the same reasoning and the same motions and ended up with the same results. Yet in each demonstration some stray rules got run at various points. In no case did these newly run rules change the system's behavior on that cycle. Instead, these rule firings resulted from a peculiarity of the relationship between the perceptual system, the dependency network, and the

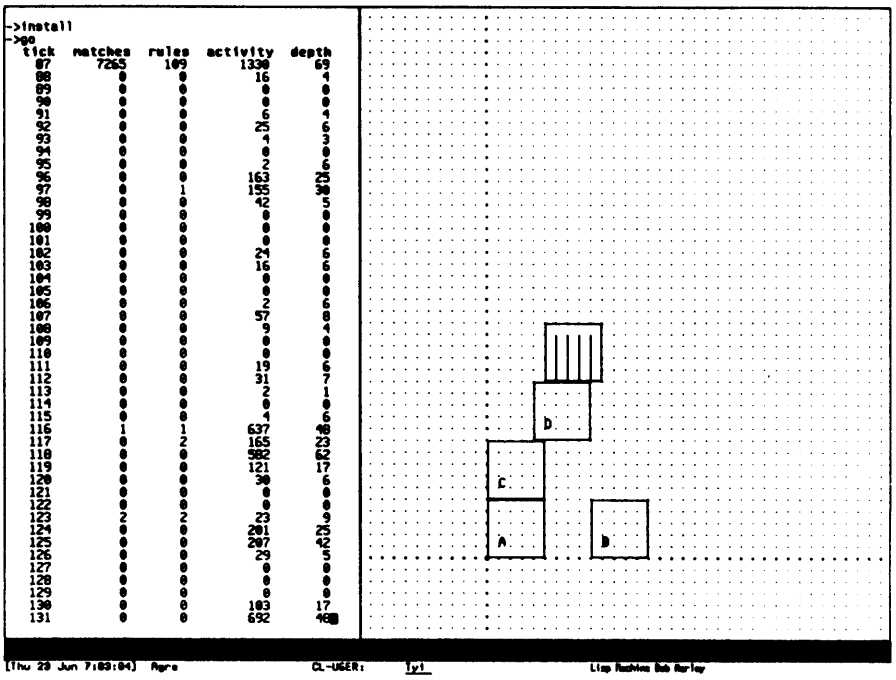

Figure B5.7(a). Having completed its two-part task, the blocks are restored to their original positions and the system is set running on the same task.

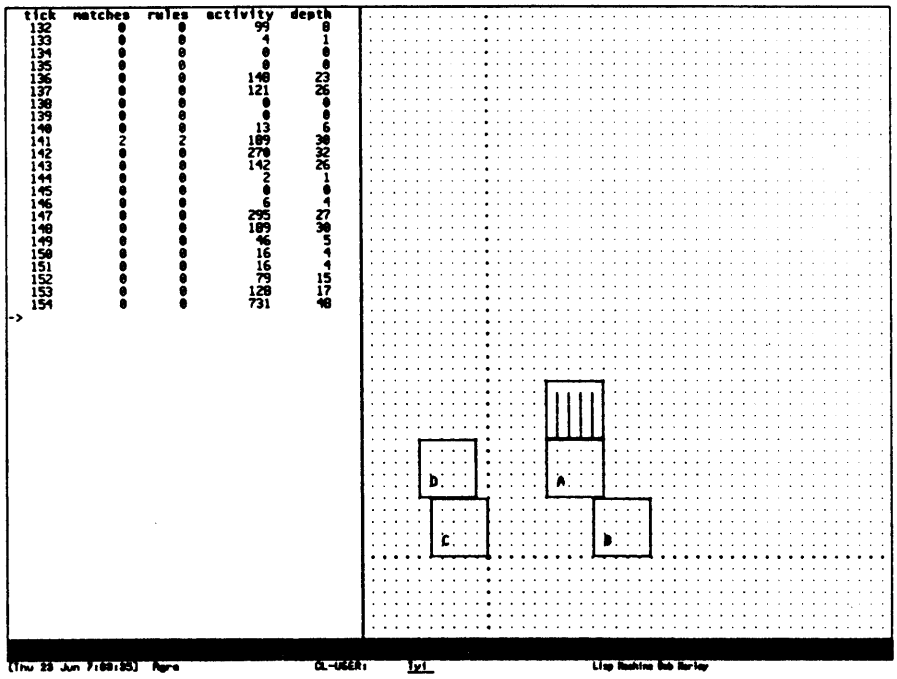

Figure B5.7(b). This time the task requires only a trivial number of rules. The levels of activity in the dependency network, though, still have a complex structure.

rule-firing mechanism. On each new cycle the perceptual system assigns new values, either IN or OUT, to the proposition corresponding to each of the primitive percepts. The blocks world is pretty stable, so the new cycle's value is usually the same as the old cycle's value. When it isn't, the change's consequences ripple through the dependency network. This network activity sometimes causes new rules to fire.

While the perceptual propositions are still being updated, though, the central system is effectively being told a false, or even inconsistent, story about the state of the outside world. So long as some of the perceptual propositions have been updated and others haven't, rules are going to fire in an attempt to pursue the current goal in the improperly specified rule state. The central system will restore itself to a consistent state once the perceptual propositions have all been updated, but only after a certain amount of extraneous activity. As far as I can tell, all of the rule firings during the second and seventh demonstrations fall in this class.

It is impossible to avoid this problem completely. Since I don't believe that people have rule systems in their heads, any attempt at alleviating the problem would only be an engineering curiosity. Some important issues would, however, arise in such a project. One is that it is unrealistic to hold the system still on cycles when hundreds of rules need to run. The whole system would be more honest if the dependency system really ran at a fixed real-time rate, occasionally letting the rule system fall behind. Unless the goal is plain engineering, though, research into such matters should wait until their connection to dynamic issues is clarified.

Let us look once again at the "depth" numbers for the sixth and seventh demonstrations. Recall that the "depth" measures the longest causal chain in the just-completed cycle's modifications to the dependency network. In discussing the first demonstration we saw that these numbers are particularly large when the system is either switching from one subgoal to another or performing a complicated argument in a difficult situation. The deeper the change in the goal hierarchy, the deeper the network activity is likely to be. The reason for this is simple. The top-level reasoning about breaking the compound goal `(and (on d c) (on a b))` into two subgoals `(on d c)` and `(on a b)`, comes IN on the first cycle and remains IN through the entire process, finally going OUT on the final cycle once the compound goal is finally achieved. The reasoning for each goal in the hierarchy forms part of the support for the reasoning for its subgoals. The depth numbers are particularly high on the first and last cycles because the reasoning for the entire goal hierarchy is going IN and OUT. In general, when the system moves from one subgoal to another, the depth numbers will be proportional to the depth of those subgoals in the system's current goal hierarchy. Thus on cycles 48, 50, and 63, when the system switches back and forth between its first and second major subtasks, the depths are 62, 62, and 48. (The two 62's are likely to be a coincidence. Only the rough order of the numbers is significant.) As the intermediate goals switch back and forth the depth numbers assume intermediate values.

B5f Conclusion

The goal of the running argument system was to do technical justice to three slogans, that everyday activity necessitates "knowing what you're doing," that it is a matter of "continually redeciding what to do," and that it is "mostly routine" in its nature. In order for the system to approximate these ideals, it somehow had to produce the effect of a huge, involved decision process on every cycle of a fairly rapid clock. The system proposed to achieve this effect by maintaining dependencies on all of its novel items of reasoning (rule firings). As a narrow technical matter, this technique is certainly a success.

As we have argued, though, maintaining dependencies is only a sufficient help if two conditions hold. First, most everything you do must be something you have done before. Although it is difficult to evaluate such a broad proposition in such a narrow and artificial domain as blocks world, I have argued (or, more honestly, posited) that this first condition is a property of the everyday activity of human beings.

Second, dependency records must transfer to a sufficiently broad range of future situations. This second condition has been the principal focus of the analyses in this chapter. As we have seen, the results have been equivocal. On one hand, a dependency record generated in one situation cannot help but transfer to a broad class of other situations, for all the reasons discussed in Chapter B2. The exact patterns of transfer will depend on the kinds of arguments and goals that drive the system and on the properties of particular domains. Unfortunately, though, the system I have described does not exhibit nearly enough transfer among situations involving different individuals. While it may not be clear exactly which analogies the system ought to be exhibiting, we have seen some clear-cut cases where it fails.

Although I have not dwelt on the point, the system also fails to exhibit another large and important class of transfer dynamics. Recall that the whole point of the transfer dynamics is learning. Chapter B2 explained the transfer dynamics using the slogan that there are fewer reasons than causes. New insights about block-stacking or cooking or driving depend on fortuitous circumstances, but once one has encapsulated an insight into a dependency record, it will be available whenever it applies. An insight might transfer to a different activity or to a different place in subsequent instances of the same activity. If a system's performance on a task has room for improvement, the dependencies ought to pick up any insights into the difficulty and transfer them to the moment when the system can use them to act differently. The running argument system was only intended as a model of routine activity and was *not* designed to explain anything about learning. Still, it will be interesting to analyze why, even though the dependencies are working as advertised, the system always performs exactly the same actions no matter how many times it is assigned a given task.

Yet another failure, already noted in the context of the sixth demonstration of this chapter, is that the system requires too much laborious rule writing for making arbitrary choices. I changed the meaning of conjunctive goals in order to avoid writing these rules

and I expect that such constrictions would plague a rule-writer in any domain where the system was not so clearly focused as it is in blocks world.

I see these three problems—failure of transfer by analogy, failure of transfer learning, and the difficulty of making arbitrary choices—as symptoms of a mistake I made in building the system. Specifically, something is very wrong about the system's perceptual apparatus and the way goals are specified. Perception works in this system the way it works in most Planners. On every cycle the system is given a full specification of the current world situation in terms of a set of propositions such as:

```
(on a table)
(on b a)
(left-of b c)
(above hand table)
(touching hand c)
(moving hand left)
(moving c left)
```

In particular, the system is given a situation (that is, a situation description) in which all the individuals have names. Now, it is odd to imagine that in real world situations you could expect your perceptual system to provide all that information. There is going to be an awful lot of it. All the individuals and their spatial relationships out in the room in front of me right now adds up to a lot of information. It is odder still that the situation description contains such constant symbols as A, B, C, HAND, and TABLE. What perceptual system knows what the objects it senses are called? Yet the input and output representations of almost all Planning systems have employed such names.

What really needs to be explained is why this practice has never seemed odd. One big part of the problem, I believe, lies in the pathology of blocks world. The choice of blocks world has also been standard practice in the Planning literature. A 'goal' presented to a Planner might read, `(put-on B C)`. One does not present the goal by aiming a TV camera at some blocks on the table, reaching one's hand into the scene, pointing, and saying "put this block on that block." As implausible as the `(put-on B C)` style of goal-specification might be for much of everyday life, in blocks world it does not seem so intolerable since children's toy blocks very often have large letters written on them. The diagrams in this literature depict the blocks in just this way. The block's name is not drawn outside the block and connected to it with an arrow. Instead, the diagrams put the name inside the block, as if one could read things' names off them. This convention would seem to suppress some hard work.

One might, however, argue as follows. "All of that is someone else's problem. We are simplifying the problem for ourselves by assuming that we are being given all this information. We aren't doing perception research, we are doing Planning research." This line of argument, I believe, is critically misleading. Referring to the objects in a Planner's input situation through names is a 'simplification' that actually makes things harder. One symptom of its falsehood is the way that dependencies fail to make transfers. Dependencies support transfer in a very simple way, whereas many

other projects have used much more complex machinery, such as pattern matchers and subgraph isomorphism algorithms, to either actually construct an analogy between two situations or to abstract away from the individuals of one situation to get a structure with variables in it which can then be instantiated over future situations.

One approach to a solution is the mechanism of *chunking* (Rosenbloom 1983) used in the Soar production-system architecture (Newell, Laird, and Rosenbloom 1986, Laird, Newell, and Rosenbloom 1987). Chapter C5 will describe the Soar system as a whole, Chapter C4 has compared running arguments to Soar's universal subgoaling, and Chapter B3 has already distinguished the running argument system's rule system from a production system, contrasting the so-long-as semantics of Life rules with the imperative semantics of productions. Chunking is a technique for summarizing the consequences of a collection of production firings that have collaborated to solve some problem by locating a successful path through a problem space. The summary takes the form of a single large production rule, or 'chunk', that collects the productions' initial premises and asserts their final consequences. Chunking is not intended to enable the system to solve any new problems, only to save it effort—albeit an often considerable amount of effort—when it faces similar problems in the future.

Chunking and dependency maintenance are related concepts. The process of building the chunk is very similar to tracing the dependencies of the productions' consequences. Like dependency maintenance, it is a learning scheme that works in the background without requiring the agent to deliberately go out and 'learn' things. Similar transfer effects appear (Laird, Rosenbloom, and Newell 1984). Whereas a simple dependency system such as that of the running argument system need only record the dependencies in the form of digital circuitry (whether actual or simulated), the chunking mechanism actually inspects the dependency network, traversing a region of it to collect its premises. A chunk effectively connects these premises directly to their consequences, skipping the intermediate layers of network structure. A system that relies on its dependency network to recapitulate old lines of reasoning need not bother short-circuiting these intermediate layers because the processing in each occurs in parallel and takes only the time of a gate delay. Chunking is necessary in a production system architecture because the system must invest a great deal of effort to fire the productions: matching the patterns, selecting the correct productions, calculating their consequences, and asserting them. Since the productions in Soar guide a symbolic search, summarizing their operation is all the more important since they may have been very numerous and because many of them may have had no useful consequences as the system explored false paths in the search space.

Chunking faces a number of difficulties. One is that it is not always safe to collapse the internal structure of nonmonotonic reasoning, lest some novel assertion come along to undo an intermediate deduction. The system must somehow recover from the resulting overgeneralization. A second problem is that the chunks themselves tend to be quite large. A chunk may summarize dozens of productions into one, but the savings will not be proportional since that one will be expensive to use (Tambe and Newell 1988).

Because it can be computationally complex to match productions with many clauses against a database, the Soar project is currently exploring schemes for restricting the expressive power of the production language (especially by requiring that a variable only appear once in the left hand side of any given production), hoping thereby to reduce the computational load without crippling programmers (Rosenbloom, personal communication).

As we will see in future chapters, a cleaner and more elegant solution lies in an account of representation that is much more suitable for an embodied agent that must interact with actual concrete materials. Instead of starting from a notion of objective individuals, it starts from a notion of focus and of the causal relationships between an agent and objects in its environment. The agent represents the objects not in terms of their objective identities but in terms of their indexical and functional relationships to the agent's body and ongoing projects. Such an account of representation has many virtues compared to conventional objective accounts. For example, it does not involve variables and is thus far more compatible with simple central system machinery and with the dependency model of routine evolution.

The notion of focus is critical. The running argument system has no focus. Its perceptual system delivers it a complete world model, as if its world were actually located in its own head. Its knowledge about the world is represented completely independently of its current goals, its current practicalities of perception and motor control, and indeed even of its existence. So far as the perceptual system is concerned, the system as a whole is always involved equally with every individual in the world, every property of these objects, and every relation among them.

This notion of focus offers a resolution to the problem of arbitration. There are usually many things one could be doing. Often a major reason why you choose one of them is that it is the first one you laid eyes on. Or perhaps it was the one that was ready to hand. When it doesn't matter very much what you choose, these arbitration schemes are as good as any. Usually one needn't even become aware of the existence of a choice.

The notion of focus also helps in understanding the running arguments system's failure to alter its behavior through the predicted transfer dynamics. Dependencies lead to routine evolution because insights about one's activities have fewer reasons than causes. But the running argument system, courtesy of its always-updated world model, always automatically knows everything there is to know about its world. It can't discover very much because the only new facts to discover are quite complex. The system's failure to evolve, of course, has other causes as well. For example, it doesn't support any sort of statistical induction.

Blocks world also tends to reinforce mistaken views of representation insofar as all blocks look alike. The blocks on a blocks-world table, unlike the tools and materials of most concrete activities, don't lend themselves to readily perceptible functional distinctions. A spatula plainly affords pancake-flipping, a hammer plainly affords nail-pounding, and your hand plainly affords all manner of things once it is suitably cupped

or flattened or clenched. Each of them carries more than enough information on it to deduce its relevance to your current activity. Blocks-world blocks, by contrast, are simple bleak squares with letters in them. All blocks look alike, regardless of their functional roles. Blocks certainly afford grabbing and stacking. But if someone says "please stack block A on block B," then nothing about either block will signal its role as the-block-to-stack or the-block-to-stack-it-onto. Lacking meaningful cues, one has no choice but to memorize arbitrary names.

There is a valuable lesson here. The domains chosen by AI researchers are often symptoms of the shortcomings of their technology. They choose the domain that makes their technology look the most obvious. This need not reflect bad faith; the people who introduced blocks world into Planning research did so because it was a simple place to demonstrate some complex forms of reasoning about subtask ordering. Once blocks world was written into textbooks and taught to students, many of its assumptions, such as the availability of a world model in which objects are automatically labeled with their names, became invisible. It is valuable to critically 'read' a domain and ask what biases it has and how it is atypical of human activities. It helps to go experience the domain yourself and compare it to the task being posed to programs. All too often you'll find that the difficult technical aspect of a program results from a failure of the program's task to correspond to any real task. I interpret the technical complexities of pattern matching and subgraph isomorphism detection as symptoms of ways in which domains have been accidentally falsified. Perhaps the skill of making and testing such interpretations can help get at the essence of activity in the world.

Part C

Deictic representation

Chapter C1

Context and summary

> **deic·tic** \'dīk-tik, 'dāk-; dē-'ik-\ *adj* [Gk *deiktikos*, fr. *deiktos*, verbal of *deiknynai* to show] : showing or pointing out directly ⟨the words *this*, *that*, and *those* have a ~ function⟩
>
> Webster's New Collegiate Dictionary, eighth edition.

In its search for interactionist alternatives to the mentalist concept of planning, Part B has introduced and demonstrated the concept of running arguments. Running arguments shift the emphasis in computational theories of action from anticipation through plan-construction to improvisation through a continual redecision about what to do now. Instead of relying on difficult hypothetical calculations involving world models and simulation, a running argument relies on all the information and resources available to an agent as it is actually at work in the world. While improvisation does not rule out trying to anticipate the future, it may relieve the need for cumbersome machinery for this purpose. We cannot know such things for certain so long as we lack a thorough dynamic analysis of such matters as mistakes, trouble, transfer, and cultural support for learning and action. Nonetheless, arguments from informal observation, computational experiments, and first principles have offered important guidance.

Experiments with running arguments, though, have found them hampered by an inappropriately mentalist theory of representation. Analysis of these experiments provisionally ascribed the difficulty to its use of constant symbols to name objects and the resulting overhead of using variables to represent information abstractly and allow it to transfer to a sufficiently wide variety of other situations. The principal work of Part C is to introduce a new, interactionist theory of representation called *deictic representation.* Not only does deictic representation fit better with an improvisational theory of action than objective representation, it is also compatible with much simpler machinery. This is a subtle fact, though, and we must approach it with some care.

Insofar as issues of representation are close to the heart of human existence, replacing one's theory of representation is like replacing one's heart. Representational theories, like hearts, are connected to a great many other things, and all of these connections

require careful attention. Chapter C2 takes up a series of such questions from the standpoint of the principle of machinery parsimony. What technical difficulties arise if we adopt a policy, such as that preached by the connectionists, of sticking with simple machinery and inventing new concepts rather than proliferating complicated machinery to implement old concepts? Starting from my thesis that the proper design of machinery begins with careful analysis of dynamics, Chapter C2 demonstrates how poorly the concepts of control, datastructures, and variables fare when we insist on starting with parsimonious theories of machinery. The tension between conventional technical ideas and simple machinery, though, is not cause for alarm. Instead, it provides a long-overdue occasion for reopening the questions which the conventional ideas were supposed to answer. In each case, new interactionist ideas, many of them admittedly poorly worked out as yet, offer promising alternatives to the conventional ideas.

Once Chapter C2 has cleared some conceptual ground, Chapter C3 introduces the notion of deictic representation. Whereas an objective representation scheme posits structures that resemble things in the outside world, a deictic representation scheme posits recurring forms of causal connection between an agent and *entities* in its surroundings. A typical entity might be *the-car-I-am-passing*. An agent will interact with *the-car-I-am-passing* by registering *aspects* of it such as *the-car-I-am-passing-is-signalling-a-turn* and basing its decisions about action on their current values.

Deictic representation resolves most of the difficulties I reported in Chapter B5 concerning my experience with the running argument system. A system employing deictic representation can be built with simple machinery because it can achieve abstraction without recourse to pattern matching and variable binding. Instead, deictic representation leads to *passive abstraction*. A deictic representation scheme represents objects in terms of their relationship to the agent and their role in the agent's projects, not in terms of their objective identity. As a consequence, a deictic representation will apply equally well to any objects that play the same role in the agent's current activities without requiring the agent to form an explicit generalization of it.

Chapter C3 then demonstrates deictic representation in the context of a computer program called Pengi that plays, in simulation, a video game called Pengo. Pengi's architecture consists of a moderately realistic visual system and a central system made of simple digital logic. Pengi's architecture can be simple because it is informed by an understanding of the dynamics of routine activity in general and video game playing in particular. In particular, Pengi's operation, unlike that of a conventional Planning system, involves a constant and very tight interaction between its visual system and central system, and between the agent as a whole and its environment.

Chapter C4 describes Pengi in detail. Most of this description concerns Pengi's visual system. Chapter A3 has already mentioned the central idea of this visual system, an idea resembling Ullman's notion of visual routines. Because Pengi depends so heavily on vision and because a Pengo screen is such a visually busy place, the details of the particular visual operators are heavily constrained. (Most likely they are *too* heavily constrained, given that playing video games seems to require greater skill in the hurried

use of one's visual system than just about any other common activity.)

Pengo bears only a glancing resemblance to any activity in the physical world. I do not mean to suggest that Pengi's ability to play Pengo proves anything about other activities. Pengi merely illustrates a number of principles—machinery parsimony, running arguments, deictic representation, visual routines, simple central system machinery, and so forth—that I originally arrived at through my studies of everyday activities and my experiences with the running argument system, long before Chapman suggested the video-game domain and began building Pengi. (Although the program is our joint work and principally Chapman's, the description and interpretations of it in Chapters C3 and C4 are my own responsibility.)

In building Pengi, we did not attempt attempt to model the transfer dynamics or any other learning phenomena. The dynamics of learning to play Pengo are still obscure to us, more so than those of most everyday activities. Future work probably should not investigate learning in Pengo. Instead it should concentrate on activities like making breakfast and driving to work. Any understanding that results should be illustrated in some more appropriate domain.

Pengi invites comparison to several existing projects. These include the classical Planning literature, a number of extended Planning schemes, the various architectures based on production systems including ACT* and SOAR, situated automata, and the MIT mobile robots. Chapter C5 discusses these projects in turn. Although many specific issues arise, a few themes predominate. One of them is the necessity of basing any theory of action on an account of the dynamics of whatever form of activity one seeks to explain. This is not to claim that I have a complete theory of activity, only that serious traps await anyone who doesn't at least try to make one. A second recurring theme is that a theory of cognition or of knowledge is not a theory of action. Focusing on action rather than on putatively mental operations drags an agent's involvement in the outside world to center stage where it belongs.

Chapter C6 concludes the report by telling some long stories to illustrate in detail the place of deictic representation in everyday life as a whole. All of these stories take place in an office and a laboratory at the Robotics Laboratory of the University of Oxford. They describe some of the ways in which these workspaces organized my activities within them and served as a background for my experience of those activities.

Some advice for the reader.

Chapter C2 consists of extended conceptual discussions that will not be to everyone's taste. It can be safely skipped on a first reading. Still, it is easy to underestimate the importance of digging up the assumptions and prejudices behind conventional computational ideas. A proper revision of computational theory from mentalist to interactionist premises requires some such conceptual exhumation sooner or later.

Chapter C3 is an approximately self-contained introduction to deictic representation and overview of Pengi. It concludes by answering a long list of objections that Pengi and

the larger project of which it is a part have routinely provoked in my presentations to technical audiences. Everyone should read this entire chapter their first time through.

Chapter C4 is a detailed, relatively formal description of Pengi and its workings. It describes Pengi's visual system at some length and explains how we went about wiring its central system circuitry. Aside from the material in Chapter C3, the only background it requires is some exposure to the logic design methods used in conventional computer architectures.

Chapter C6 tells stories. Since most of the stories' points correspond to nothing I have implemented, one might consider it a 'Future Work' chapter. Because of the scope and difficulty of its ideas, this chapter is much more careful in presenting its narrations than previous chapters have been. Even so, readers with a background in Continental philosophy or sociology will find that the theoretical apparatus has been considerably watered down for a technical audience.

Chapter C2

On connectionism

C2a Context and summary

After a long struggle, one day I had to admit that the ideas in Part B made me a connectionist. A connectionist is a computational psychologist who works from the hypothesis that the machinery in our heads is a connection network, that is, a large collection of simple, uniform 'nodes' interacting through a fixed arrangement of 'connections'. This is quite a drastic assumption, given that AI programming has heretofore made heavy use of transient, easily reconfigurable networks of datastructures joined by pointers. For many, connection networks are appealing because they seem 'brain-like'. For me though, connection networks are merely a logical starting place. Parsimony suggests exhausting the dynamic possibilities of simple machinery before positing anything more sophisticated. What sorts of dynamics do agents with connectionist machinery get themselves into?

Unfortunately, this is a hard question to answer because our ways of thinking about computation are shot through with unarticulated assumptions from conventional hardware and software practice. Previous chapters have begun the project of critically re-examining conventional ideas about computation. This chapter carries on, using the difficulty of reconstructing conventional methods in connection networks to help focus the search for alternative methods. Section C2b expands on this strategy by considering the slogan of 'software', through which computational practice has become increasingly 'abstracted' (*i.e.*, detached) from the 'details' (*i.e.*, the fundamental nature) of the physical realization of its computations. Subsequent sections consider particular issues:

Section C2c concerns the notion of 'control' as a metaphor for the organization of a computation. Both 'control' and the related metaphor of 'search' originate in a version of AI whose prototype of human activity is detached cogitation about abstract puzzles. The combinatorial arbitrariness over which puzzle-solving algorithms must struggle to maintain control is almost entirely absent from concrete activity. This, together with our partial understanding of the dynamics of concrete activity, makes it plausible to excuse the difficulty of building search spaces in connection networks.

Section C2d concerns the notion of 'datastructures' as an account of activity involving structured entities like sentences, plans, and lists. Datastructures are hard to implement in connection networks because pointers let one connect anything to anything else at any time. I argue that this flexibility is excessive. Also plausible is an alternative (though still sketchy) account that emphasizes so-called 'internal language' and interactions with structured physical artifacts like signs, instructions, and shopping lists.

Section C2e concerns the notion of 'variables' as an account of agents' abilities to abstract their knowledge away from particular circumstances. There is a metaphorical tension between the notion of 'variable binding' and the fixed configuration of connection networks. And indeed, variable binding has proven cumbersome to implement in connection networks, even in restricted cases. The problem originates in a mistaken ontology. Most existing semantic theories, for example the usual model theory of first-order logic, posit a relationship of 'reference' between constant symbols in a representation and objectively individuated objects in the world (or the model). This section discusses some of the difficulties with this account; Chapter C3 presents an alternative account based on 'aspects' of indexically and functionally individuated 'entities'. Aspects are compatible with a wide range of connectionist proposals, including accounts that (unlike my own) take some position about the nature of state.

A central, recurring theme of this chapter concerns the *temporality* implicit in models of computation. An agent's machinery, I have been suggesting, is principally oriented toward its continual interaction with the world. If so, one must evaluate a theory of machinery according to a model of computation whose notion of time is the real time of the agent's concrete activity. Models of computation have rarely had much to do with the outside world, but ideas about programming have moved steadily even further away from this ideal, to the point of celebrating models (like logic programming and functional programming) that have no temporality at all. A connection network can operate in its owner's 'real time', provided it has a continual causal connection to its owner's world.

C2b Connectionism reminds us of what's important

Why can't our computers figure out for themselves what to do? This chapter considers two answers to this question. One answer, as Chapter B4 began explaining, is that computers haven't been allowed to participate in concrete situations in the world. They haven't got anything interesting to do. This chapter begins explaining a second, closely related answer. 'Software engineering' has been overwhelmingly preoccupied with getting systems to work at all. This is perfectly understandable. But it has meant that computer technology has consistently suppressed most of the constraints inherent in a device's physical realization. This trend is most obvious in the predominance of serial computers. Seriality simplifies programming at the expense of an extremely unnatural physical realization.

The notion of software also tends to suppress issues of physical realization. The purpose of a programming language, it is often said, is to 'abstract away from the details of the hardware' by affording the programmer the illusion of working on a simplified 'virtual machine'. This 'abstraction' is often extreme, letting programmers make ready use of such facilities as recursion, backtracking, symbolic programming, dynamic storage allocation, and virtual memory. While extraordinarily flexible, these capabilities have a price. Both linguistic abstractions and serial computer architectures themselves tend to obscure just how much overhead they involve. While we have many ways of talking about their 'power', we have few ways of talking about their cost. As evidence for the view that this 'power' is illusory, I submit that large AI systems that 'exploit' it are so drastically underconstrained that they regularly get lost in a 'search space' of indistinguishable options. The task of keeping the space of options from getting out of hand is often reified as the 'control problem'. This 'problem' is just as hard as the entire 'AI problem' and much harder to think about. As Section C2c suggests, 'control' should be understood as a symptom, not a research project in itself.

Two schools of AI research have given a central place to issues of physical realization. One is Minsky's remarkable 'Society of Mind' project (1986). The other is a confederation of researchers calling themselves 'connectionists' (Feldman and Ballard 1982, Feldman 1985, Hinton 1981, McClelland and Rumelhart 1981, Rumelhart, Hinton, and Williams 1986, *et al*). The connectionist movement is united by an admirable but troublingly vague desire to make 'neurophysiologically plausible' theories of cognitive machinery. Connectionists intend their models to be as easily physically realized as possible. These models involve large numbers of simple, uniform 'nodes' interacting through a fixed arrangement of 'connections'. Such a 'connection network' can be drawn as a graph. Ongoing debates concern whether the graph is directed, how many kinds of nodes there are, how network behaves over time, and so forth.

What's really staggering is the amount of conventional 'programming technology' that has no practical analog in connectionist systems. Section C2d will consider the matter of pointers and datastructures. Despite some heroic attempts (*e.g.*, Hinton and Touretzky 1985, Touretzky 1986), it seems impractical to use the static connectivity of a connection network to provide the dynamic reconfigurability required by symbolic programming as currently understood. Section C2e will consider the more subtle matter of variables and constants. The possibility of binding any of several variables to any of several values would seem to require hardware proportional to the number of possible bindings. In each case, it seems to require considerable overhead to implement conventional methods in connection networks. I can think of several possible responses to this situation.

- One might use these prospective difficulties to justify dismissing connectionism as an arbitrary and unmotivated constraint.

- One might be indifferent to connectionist research, considering that it operates at 'too low a level of abstraction'. On this view, connectionism might be a harmless

pastime, but at best it can only reconstruct what we otherwise know at more appropriate levels. I will argue that this attitude—exhibited to a remarkable degree by some connectionists—is hubris.

- One might wait for unforeseen practical techniques for implementing analogs of conventional programming technology in connection networks. These techniques would certainly be an important discovery, but I will argue that we probably wouldn't want them anyway.

- One might decide that all the overhead is worth paying. Having decided this, one might develop a connectionist technology based on standardized building blocks and higher levels of abstraction. It's at least plausible. If a million connections don't sound like much to you then it's a live option, though some care will be required to achieve acceptable rates of convergence.

- One might look for alternatives to pointers, datastructures, variables, and constants that one can implement in a natural manner in small numbers of connections. This is the approach of this chapter. I will present such alternatives and survey their consequences and limitations.

It is certainly too early to choose among these alternatives. Clearly, though, embracing connectionism entails a lot of work. For many, a choice among these alternatives will depend on intuitions about whether all the effort will be well spent. One might well ask, isn't it backwards to let decisions about machinery drive research? And why exactly *connectionist* machinery? In designing connection networks, isn't one just wandering in mechanism space? Even *if* the models can be made to do something, what will be learned? Everyone needs answers to these questions. I have two. One is that the constraints of connectionism, by forcing attention to issues of physical realization, encourage a useful conceptual housecleaning. When long-accepted ways of defining AI's research tasks lead to unappealing technical aporia, one is encouraged to reopen old questions. Why did we want our agents to have pointers? And why variables? And why control structures? In retrospect, many traditional ideas will seem to depend on accidents of technological history, such as the serial operation and primitive 'I/O' facilities of early computers. Using accidents to define the terms of research, while perhaps inevitable and understandable as a historical matter, is nonetheless asking for trouble. It is too easy to treat the symptoms of such a mistake as natural and necessary, as topics of research in themselves. An artificially constrained new beginning can focus the project of exposing these accidents *as* accidents. The slogan is,

> Connectionism doesn't cause problems, it reminds us of them.

A second answer is that concern with machinery is only an *invitation* to wander in mechanism space, not an *order*. What we'd like is some principled account of what connectionist machinery is and isn't 'good at'. The last way to achieve this is to consider a

connection network in isolation, like a kitty-cat with electrodes stuck in its head. Well before this detailed stage we should ask basic questions about machinery and dynamics. What are the dynamics of everyday activity and how appropriate is connectionist machinery for an agent whose life is organized that way? By frustrating the habit of throwing sophisticated machinery at every new problem, connectionism compels us to move the focus of our research from detached cognition to situated activity, from internal mental processes to the dynamics of an agent's interaction with the world. Critically, the resulting insights are largely independent of the particular constraints of connectionism.

C2c Beyond 'control'

Though I don't recall ever seeing it written, I have often heard it said that "all interesting AI problems are intractable." I can only hope this is intended in the technical, capital-letters sense. When a Problem is intractable, the best possible algorithm typically takes the form of a nondeterministic 'search' through the whole space of possible solutions. Many early AI people appear to have been happy to resign themselves to this sort of algorithm. Since any Problem can be formalized in this way, 'search' seemed like a powerful unifying theme. Artificial Intelligence would be achieved as soon as someone resolved the detail of how to 'control' this search. As a side-effect of this enthusiasm, within the AI community the word 'search' quickly came to name a mental process rather than an activity in the world.

It is only a mild exaggeration to categorize present-day AI schools as deriving from the history of various reactions to the difficulties of search control.

- Some consider search control a problem for someone else, or for the future. These people are happy to make 'competence theories', especially in first-order logic and its modal extensions, without any concern for whether they could ever be tractably used.

- Some investigate the properties of fixed search-control policies. This approach, though far from extinct, is not currently ascendant.

- Some observe that little or no search is required if one chooses the right representation and pose AI as the science of good representation. Since good representations are domain-specific, this sort of research tends to focus on individual problems and postpone research on AI as a general matter.

- Some observe that little or no search is required if one has enough 'knowledge' and pose AI as the accumulation of knowledge. This attitude characterizes 'expert systems' research. It works to exactly the extent to which you can assimilate everything to 'knowledge'.

- Some focus on the widespread observation that 'search control' is every bit as hard as AI in general. This suggests applying the whole range of AI methods at the 'meta level', so they set off to investigate 'meta-level architectures'.

Many schools take elements from more than one of these responses. All five have had their day at CMU, where the slogan of 'search' is most prevalent. All five have their bit of truth. My purpose here is neither to attack them nor to choose among them. Instead, let us consider the metaphorical presuppositions behind 'search' and, in particular, behind 'control'. The issue is topical because facilities for building and exploring descriptions of search spaces are singularly difficult to implement in connection networks. Does search make a useful central concept for understanding everyday routine activity? I contend not.

What is the word 'search' telling us? In a 'search' one is looking around in some territory, often known as a 'space', for some object, starting at a definite location. The searcher might keep some map of where it has been, but it only knows about the regions it hasn't visited in the most general terms. A search conducted by an algorithm in an abstract space is different from a search conducted by an agent in the physical world. Abstract search spaces are strange places with little sense of temporality, continuity, or familiarity. Abstract searchers typically don't pay any price to travel long distances in the known regions of the search space. A search algorithm rarely has any business in its space except finding its object; the space doesn't change and the object doesn't move. Most importantly, though, abstract searches typically take place in vast or infinite territories. Only rarely is a search space entirely laid out in a machine. Instead, descriptions of the locations, called nodes, are built as they are needed. It is this building that is difficult to implement in a connection network.

A search algorithm has two modules. One module incrementally builds the space by 'opening' a node; the other 'controls' the search by deciding which nodes to open. Speaking of a search 'algorithm' emphasizes the seriality of the search metaphor: you can only look one place at a time. A further system of metaphors addresses this modularity. A 'control scheme' must take immense care because, in most search spaces, an exponential amount of territory unfolds with growing distance of the starting point. Careless searching results in an 'exponential explosion'; one imagines the control scheme working to keep this ever-incipient explosion 'under control'.

A search process is like a nuclear power plant, where 'control rods' govern an inherently exponential process so as to avoid an explosion. Each case involves an intervention from outside an existing process that regulates its level of activity. But there is a critical difference between the two cases. Whereas a nuclear plant only cares about the amount of activity, a search algorithm also cares about *which* activity is permitted. A control scheme needs foresight; it has to be ruthlessly selective about which directions to search without knowing where the object is. It has always been extremely difficult to reconcile these conflicting demands. Parallel search methods can help, but the constant factor they can provide is nothing in the face of an exponential explosion.

The metaphors of AI's search methods are mentalistic. Search happens inside the

computer or inside the head. Meanwhile, what's happening in the world outside? Search methods have generally been applied to difficult, abstract intellectual tasks which let us imagine the world waiting patiently for an answer. They have not often been applied to more participatory activities in which the temporality of the agent's being is stereotypically at issue. As a consequence, the temporality of a search process is oddly formless. It doesn't matter in which order the agent opens the nodes, nor how many it opens at once, nor (within some large factor) how many it opens in total. And this formlessness runs deeper. The explosion of a search process is most often identified as a 'combinatorial' explosion. This means that, at every step, the control scheme faces a set of formally indistinguishable options with little benefit of experience and no recourse to the world outside. When combinatorial searches can be controlled, it is because these arbitrary choices can be constrained using information about the domain or about the specific situation occasioning the search.

It has seemed natural to apply search methods to difficult, abstract intellectual tasks, for two reasons. First, these tasks are stereotypically non-routine. By putting a premium on novel thinking, they justify metaphors of exploration and construction. Second, these tasks are stereotypically non-concrete. By putting a premium on detached cogitation, they justify metaphors of autotelic processes unconnected with the world outside. In each case I say 'stereotypically' because these are idealizations. In real life, much is routine in, say, an experienced engineer's problem-solving. Likewise, most people who prove theorems and solve puzzles do so on scratch paper, often interacting with the paper in sophisticated ways. Nonetheless, the temporal formlessness of abstract search processes makes them so open-endedly Protean that it almost impossible to argue against them. But why start out by studying difficult, unnatural, and culturally specific activities? If someone tries applying some method to such an activity and never quite succeeds, little has been learned.

Concrete activity can be difficult, but it certainly does not share the temporal formlessness of abstract search. The temporal organization of concrete activity is highly constrained by the practicalities of an agent's interaction with its workplaces, materials, and equipment. This constraint, the inherent orderliness of concrete activity, is certainly not complete. But it is so extensive that it challenges the whole metaphor of 'control'.

What is the word 'control' telling us? The metaphorical likeness between abstract search and a nuclear plant has already suggested one element: control stands outside a process and attempts to limit it. The word 'control' entered English from the French phrase *contre-rôle*, literally counter-ledger, which referred to a feudal practice of monitoring the fiscal affairs of an estate through a second, duplicate ledger. In general, control connotes an oversight so close as to be unnatural. To speak of something as 'under control' is equally to invoke a latent potential for getting 'out of control'. The extensive black markets and large underground cultures of Communist countries speak of the illusory nature of total political control. People who are 'controlling' or 'controlled' are artificially rigid in their dealings with others or with themselves, presumably be-

cause they feel they have to be. The coiled spring one must control is an aberration in the first place.

This analysis suggests a diagnosis of search-control disorders. Few search methods have been informed by the orderliness of routine activity, whether by continually looking to the world for guidance or by learning from long incremental experience (Soar is an exception). Instead they have relied on control schemes that attempt to reproduce this orderliness through artificially explicit, centrally applied policies. But life is not like this. Your life is 'under control' not through any artificial 'self-control' but because you have found a way of smoothly participating in the network of criss-crossing dynamic patterns that make up everyday life. An important question for dynamic theory is, how it is that most people can feel more or less comfortably involved in the dynamics of everyday life?

These considerations offer a new perspective on the nature of parallelism in machinery. We saw that the mentalist solipsism of abstract search produces a formless, uncontrollable proliferation. More generally, mentalist metaphors drastically underconstrain our attempts to choose among different styles of parallelism. If a computation is metaphorically disconnected from any 'real time', all temporal organization is a misfortune to be dismissed as a variety of archaic seriality. If all forms of parallelism are equally desirable, a process is judged solely on how close it comes to finishing in unit time. If there are activities for which the temporality of the outside world is truly superfluous, then this makes sense. But everyday routine activity is not like this. Once we recognize that our continual decisions about action must take continual account of the outside world, we can focus the question: what sort of parallel machinery best supports this moment-to-moment assessment and decision?

Obviously this is a big question. But, just as obviously, a connection network has this much to recommend it: when changes in the world cause its inputs to change, it devotes massive parallelism to adjusting to the new information. The new information might be significant in millions of different ways, and the network checks every possibility at once. We have already seen many manifestations of this sort of noticing; we will see many more. Indeed, without some sophistication, connection networks are *too* responsive to their inputs. Future work on connection networks must determine what provisions to make for maintaining state that is neither directly wired into the connections nor directly available from the periphery or the world.

(For further discussion of the relationship between connectionism and the dynamics of serial order in organized activity see Jordan 1986 and Jordan and Rosenbaum 1988.)

C2d Beyond datastructures

The history of programming is largely the history of the index register. Using index registers, language designers have gradually pulled loose from the constraints of fixed address assignment by devising ever more extensive standardized schemes of indirect reference. The resulting abstractions have grown steadily more extreme, but my argu-

ment centers on two innovations of the late 1950s, stack frames and pointers. The next section concerns the consequences of the generalization of variable semantics permitted by stack frames; both Algol and Lisp use stack frames to determine the reference of local variables relative to a particular call on a procedure, even if several calls are active at once. This section concerns pointers, datastructures, and dynamic storage allocation. A pointer to an object in memory is simply a binary representation of the object's address. A process can gain access to an object by obtaining a pointer to it. Most commonly these objects are *datastructures.* (To simplify discussion, I will restrict the term 'datastructure' to mean a dynamically allocated record structure that processes can inspect and modify through a pointer.) Large AI systems are usually made mostly or entirely of datastructures.

Given the prevalence of datastructures in AI models, it is tempting to ask whether people have datastructures in their heads. The question is made topical by the severe tension between the transience and reconfigurability of datastructures and the static configurations of connection networks (Fodor and Pylyshyn 1988). Connectionist implementation of datastructures—at least in general, at least in any obvious way—has proven extremely difficult. On reflection, though, the question isn't very useful. Few AI models insist on being implemented with datastructures. Instead, they insist only on an abstract specification, sometimes called a 'competence model' or a 'computational theory'. Even when a model provides for structured entities like parse trees or plans, pointers and datastructures are merely implementation details. Or so it is held. Since nobody seriously believes that human memory has numerical 'addresses' with which to implement pointers, we need to consider the notion of structuredness more abstractly.

One of the principal origins of the modern-day psychology's notion of structure is Chomsky's notion of generative grammar (1965). Indeed, computational ideas of structure still closely mirror the context-free grammars of the simplest generative linguistic theories. A context-free grammar might specify

```
S → NP VP
```

meaning that a sentence is a noun phrase followed by a verb phrase—and that *any* noun phrase and *any* verb phrase can be juxtaposed to produce a grammatical sentence. A sentence is still a string of words, but only on the surface; each sentence has a recursively composite 'phrase structure', often drawn as a tree with an S at the top and individual words at the terminals. Programmers use different metaphors. Instances of S, NP, and VP might be called 'objects'; an S might be said to 'contain' an NP and a VP, or alternatively to 'point at' them.

For Chomsky, faced with the intellectual desert of behaviorist linguistics, ideas of structure were profoundly significant because of their *generativity.* If behaviorism spoke of a fixed repertoire of 'verbal behaviors', according to Chomsky a proper grammar permits the construction of all the grammatical sentences of a language. In doing so, the grammar "makes infinite use of finite means" (1965 p. *v*, quoting Humboldt). Chomsky starts from an assumption that syntax is 'autonomous', meaning that one can specify in

wholly syntactic terms the competence speakers need to relate the 'surface' and 'deep' forms of a sentence, whether to produce a sentence or to parse it. Chomsky professes agnosticism about whether these phrase structures correspond to datastructures in human heads. Still, it is undeniably attractive to interpret the observation that people can act in novel ways and invent novel ideas in terms of newly generated structures. This intuition suggests structured conceptions of AI ideas like plans and knowledge representations.

A second virtue of structures is *reconfigurability.* Chomsky observes that his tree-shaped phrase structures permit relatively accurate accounts of the forming of questions, at least compared to completely unstructured models. When "he ate a crayon" becomes "what did he eat?," Chomsky sees a structural transformation from the former sentence's phrase structure to the latter's. AI models, too, continually posit processes that modify structures, either by rearranging their elements or by replacing existing elements with new ones.

A third virtue of structures might be called *inspectability.* Structures encourage visual metaphors: a process using a structure is often said to be *looking* at it. This metaphor describes a radical separation between structures and processes: structures are passive and processes are active. Any process can look anywhere in any structure it likes. The issue of inspectability has already arisen in Chapter B2's discussion of dependencies: methods such as dependency-directed backtracking must inspect dependency networks, but the use of dependencies to recapitulate old lines of reasoning can be implemented using uninspectable combinational logic.

There is no denying the place of structure in our computational theories of psychology. And pointers provide one simple, clean way of implementing generative, reconfigurable, inspectable structures. But pointers also require machinery more sophisticated than connection networks. The critical question here is, when *must* we interpret structure in terms of something like pointers? The differences between Chomsky's and AI's structures help focus the issue. Putatively universal properties of human language strongly constrain all three aspects of linguistic structures. Early Chomskyan grammars, like the type systems of all conventional programming languages, could generate any recursively enumerable language. The generative power of more recent grammars, though, has been much more precisely circumscribed. As a result, and because linguistic structures need only be manipulated by a fixed collection of mechanisms, computational linguists can now build plausible parsers using relatively simple machinery. These results do not depend on Chomsky's particular formulations, nor do they require syntactic processing to be subserved by a dedicated peripheral module. One can exploit the natural temporality of language by working left to right through the sentence (*cf.* Marcus 1980), or spread the parsing task among the machinery implementing the lexicon entries (*cf.* (Small 1981) and (Small, Cottrell, and Shastri 1982), see also Berwick's (1983) commentary on Small *et al*).

Syntax can afford the constraint necessary to simplify its machinery, but how about Planning? The question has rarely been asked because AI's 'symbolic programming' has

always taken full advantage of the 'power' of unconstrained generativity. For symbolic programming, structures are passive and wholly inspectable. Anything can be connected to anything else at any time. Generality always has a price, and the general use of pointers in symbolic programming is no exception. But serial computers conspire with the metaphors of structure to hide this price. The von Neumann bottleneck permits a process to follow only one pointer at a time and reinforces the centralized, homuncular imagery of 'looking at' a structure. It's certainly possible to permit many processes to inspect many structures by building generalized switching networks, but it's also inordinately clumsy. This clumsiness is not a natural, normal research topic, it is a symptom.

An interactionist alternative to pointers and datastructures begins with the hope that language-use requires only relatively simple machinery. This alternative account of structure (already stated in a different, partially interactionist, form by Rumelhart, Smolensky, McClelland, and Hinton 1986) has four parts:

1. Structure-use is exceptional. For most purposes, activity is subserved by interactions among a connection network, some peripheral machinery, the body as a whole, and the outside world. In particular, plan-use is only an occasional supplement to the prior dynamics of moment-to-moment improvisation.

2. It is best not to think of 'structures' as a formal calculus of things-in-the-head at all. The prototypical structures are structured *artifacts* like maps, shopping lists, signs posted over supermarket aisles, instructions on cake mix boxes, spoken advice about testing batter, and so forth. Interactions with structured artifacts have their own complex dynamics.

3. Many cognitive skills develop in the course of interactions with structured artifacts. This process, often misleadingly called 'internalization', proceeds through a disparate collection of dynamics that gradually replace the presence of the artifacts. Their effects are haphazard and incomplete, and they generalize poorly. Above all, they rarely work by, in any sense, reproducing the structure in your head. For example, you may not need to consult a recipe once you've followed it two dozen times, but even then you'd have little chance of reproducing it verbatim (*cf.* Hutchins 1987). In general, it is extremely difficult to remember structured information by rote.

4. The principal exception is so-called 'internal speech', the capacity for which appears to arise through a complex 'internalization' of hearing oneself speak. Internal speech is unlikely to be subserved by datastructures because it is (or at least, seems to be) time-extended in the same way as ordinary speech. It is also highly elliptical and indexical in the same way as children's egocentric speech (*cf.* Vygotsky 1978) and adult conversations (*cf.* Sacks, Schegloff, and Jefferson 1978).

5. Even our interactions with external structured representations are time-extended

processes. Arraying structures in time rather than space is more compatible with simple machinery (*cf.* Elman 1988).

The mentalistic connotations of the phrases 'internalization' and 'internal speech' are a shame. Speech is the very clearest case of an activity that is indifferent to inside-outside boundaries. Even when it is formal or asymmetric, a dialog smoothly merges two 'internal' monologues.

C2e Beyond variables and constants

An account of abstraction must explain how an agent might transfer the lessons of an episode on Monday to a new situation on Tuesday. By far the most widespread account involves variables and constants. Monday's objects are named by constants; the agent abstracts by replacing these constants with variables. The resulting abstract structures embody Monday's lessons independently of Monday's particular objects. On Tuesday, the agent replaces the variables with constants naming the new objects.

How might variables be implemented in connection networks? In the simple case where variables and their values are localized in the network, some sort of crossbar must join the variables to their possible values (McClelland 1984, Blelloch 1986). Or one might allocate a separate node for every possible variable-value pair and force them to exclude one another by using some method like a 'winner take all' network. With a small number of values this is acceptable; with a large or infinite number it isn't. Unfortunately, many ranges are infinite, such as the names of possible objects or people. Algol- and Lisp-style variables are especially difficult because connection networks have no way of generating fresh copies of a structure on demand. Alternatives implementations have been scarce. Distributed representation (Hinton, McClelland, and Rumelhart 1986) can save a small constant factor at the price of some unreliability in the form of 'crosstalk' errors (Feldman 1986, Willshaw 1981). (See also Blelloch 1986, Chapman 1988.)

What is the problem? Let us consider first-order universal quantification as an account of abstraction. A universally quantified formula states that some predicate must hold true of every individual in any model of the theory. A theory can name individuals using a constant symbol; each constant symbol will refer to some individual in any model. To generalize some insight from Monday's activity, then, one replaces the constant symbols naming Monday's objects with universally quantified variables. To transfer this insight to Tuesday, one instantiates the quantified formula over symbols naming Tuesday's objects.

There is no way to enumerate the variations of this story. Can your kitchen be a model? Or is a model a mathematical object? If so, what is the relationship between a kitchen-model and a real kitchen? In any case, the picture has three players: a variable, which might be instantiated with some constants, which refer to individuals in the world. This is the classical account of representation. Almost all AI representation

schemes are instances of the classical account (*cf.* Hayes 1977). Indeed, so are almost all computer programs, insofar as their calculations concern things in the outside world. So, in particular, are most of AI's 'symbolic programming' schemes, insofar as they have any clear semantics at all.

What good is this story for an agent trying to get on in the world? It's incomplete at best. The symbols in a classical representation may *refer* to individuals in the world, but this says nothing about the agent's *causal* relationship to those individuals. The reference relationship is simply *posited.* In virtue of *what* does a reference relationship obtain? Anglophone philosophy has generally formulated this as a question of epistemology; none of the innumerable proposals are very convincing. AI systems have usually fudged the issue by permitting their primitive perceptions and actions to traffic in constant symbols. This is an all-too-easy mistake when the 'world' is simulated using a subprogram that shares constant symbols with the model. But when a Plan contains `MOVE(BLOCK-A,BLOCK-B)` as a primitive operation, something is wrong. Why is this tolerated? It is tolerated because mentalist ways of speaking hide its absurdity. (For a fine example of this absurdity, see Chapter B5.)

We have here a especially bad case of mentalism. Unless you think about it carefully, mentalism will make it plausible that acting on symbols in your head is just as good as acting on the world. The gap between reference and causality is wide. Converting constants into actions and perceptions into constants can be, and often is, hard work. It is not always possible. When it is, it regularly requires one's full intelligence. Picking up the magnetized screwdriver might mean trying both screwdrivers' tips next to a screw or figuring that it must be in the kitchen where you last used it. Picking up your water glass might mean looking at which of your neighbors at the table seems to have two. Picking out the right key might mean rubbing your thumb against them to see which has the square head.

Other cases require a tremendous amount of keeping track. It would take some work to figure out that the newspaper that's lying on your kitchen table when you get home at night is the same one you left there in the morning. As if it mattered. Of course it's easier once you know that it's the same table, but that takes work too.

What's worse, the hardest cases are often the ones that matters least. If your roommate eats the last of the corn flakes before bed and opens a new box, then you need to know that this morning's corn flakes box is `CFB-137` rather than `CFB-136`. Worse yet, if you are fishing one screw after another out of a box, they all need their own symbols. You need a symbol for every coathanger in your closet, every coin in your pocket, every can in your cupboard, every chair you ever sit in, every car you ever pass on the road, every cup you ever drink from, and every page you turn in every book you read. Symbols with meaningless numbers on them ought to suggest that meaningless distinctions are being made.

The problem, I claim, is ontological. Just about every semantic theory ever made, including the usual model theory of fopc, assumes that symbols refer to objectively individuated objects out in the world. By "objectively individuated" I mean defined

independently of the location, attitudes, projects, or even existence of the agent. On this account, the world is populated with objects. These objects were already there before you came and they'll still be there when you go. The objects and their relationships are independent of where you are and what you're trying to do. It doesn't matter whether this account is *true*, it only matters whether it is *useful*. In Chapter C3 I'll present an alternative account according to which objects are individuated indexically (according to their relationship to your body) and functionally (according to their relationship to your purposes). For the moment, though, let us consider the shortcomings of objective ontologies in more detail.

All the unnecessary distinctions required by an objective ontology make for unnecessary work. For example, it seems redundant to instantiate your universally quantified automotive knowledge for every car you pass. In AI practice, this instantiation usually occurs in a pattern matcher. Control structures based on pattern-matching have long been standard practice in AI. They were central to a long line of 'production systems' (see Section C5d); they were also central to the 'pattern-directed' systems that were common in the 1970s (Waterman and Hayes-Roth 1978). A pattern matcher applies a set of 'patterns' to the propositions in a database. The matching process is purely syntactic; it conducts no reasoning and makes no reference to the world outside. When a pattern matches, some code is executed. The system alternates between matching and execution.

The simplicity of pattern-matching algorithms is purchased at the same price as the simplicity of semantic theories based on reference. Determining whether your present circumstances fit a pattern is a fairly complicated matter in real life, but the slogan of 'pattern matching' reduces it to a simple structural comparison. This simplification doesn't eliminate any work, it only pushes it around. Before pattern-matching can help you, you have to have represented the world in the terms required by your patterns. What's missing is an account of how the propositions get into the database. The problem is hidden in puzzle-solving and other simple scratch-paper domains: it is so easy to reproduce the world in the database that there is never any need to actually look at it. In real life, by contrast, one must continually choose among a hundred ways of looking at the world.

Furthermore, systems based on pattern-matching derive an inherent seriality from the need to ration access to the pattern-matching machinery. In a production system, for example, only one production can run at a time. We have already seen that there is nothing wrong with seriality, provided it corresponds to the natural seriality imposed by the dynamics of one's activity. But the seriality of a production system is at a much finer level: the very smallest units of cognition much occur in serial, not the resulting physical actions. Running arguments do not have this restriction; all the circuitry runs all the time. The next few chapters demonstrate this in practice.

Chapter C3

Pengi

C3a Context and summary

Previous chapters have expressed in some detail my dissatisfaction with the theories of action implicit and explicit in the models of artificial intelligence, particularly the notions of world models and planning. This chapter presents some new ideas about action, beginning from a new, interactionist theory of representation called *deictic representation.*

These ideas are embodied in a computer program called Pengi that plays a video game (Agre and Chapman 1988). David Chapman and I, though principally Chapman, wrote this program as an exercise in building systems that engage in an improvisatory interaction with continually evolving surroundings. Rather than maintaining elaborate world models and constructing symbolic plans, Pengi relies heavily on its interactions with the world to organize its activity. As a consequence, Pengi could be made of very simple machinery, consisting of a moderately realistic visual system and a central system made of combinational logic. The orderliness of Pengi's activity and its effectiveness at playing the game arise not through exhaustive knowledge and simulation but through the continual interaction among its central system, its visual system, and its surroundings.

Section C3b introduces deictic representation. Chapters B5 and C2 have already discussed the need for an account of representation that does not rely on variables and constants and that can be conveniently implemented on simple, connectionist hardware. This section distinguishes deictic representation from the more conventional objective representation.

Section C3c introduces the Pengi's domain, a Lisp Machine reimplementation of a commercial video game called Pengo. It describes some of the more important aspects of the activity of playing Pengo. Finally it introduces the Pengi program itself, briefly describes its organization, and summarizes the relationships between its machinery and dynamics that later sections will discuss in more detail.

Section C3d describes some of the ways in which Pengi uses deictic representation to

play Pengo. Formulating Pengi's deictic representations depends on an understanding of the common patterns of interactions between skilled players and the game.

Section C3e describes the way in which Pengi embodies the ideal of improvisation. Its machinery engages in a continual interaction with its perceptual apparatus and with the outside world. It proceeds not by following a worked-out plan but by continually redeciding what to do based on each next moment's situation. By leaning on the world to provide for much of its representational and control needs, it avoids the complex machinery prevalent in past systems. Chapter C4 will explain in much more detail just how Pengi works.

Section C3f draws out the close relationship between the computational themes of seriality and focus. The necessity of constraining one's focus to particular items or regions or issues entails serialized interactions. Chapter C2 has already discussed this theme abstractly. In Pengi the theme arises at three levels: in the dynamic phenomena inherent in having a body, in the notion of focus realized by Pengi's visual system, and in the abstract notion of 'functional' focus that is part of deictic representation.

Section C3g addresses a number of objections that arise in discussions of Pengi. Many of these concern the precise distinctions to be drawn between Pengi and other technical ideas and projects.

C3b Deictic representation

Traditionally the study of representation in AI has been centered on issues of expressive power. Talk about these issues is organized by mentalist metaphors of inside and outside. A representational token in one's head expresses some state of affairs in the world; one manipulates and recombines these tokens in ways that, one hopes, preserve their faithfulness to outside circumstances. Competence at worldly activities is founded upon the accuracy of correspondences between structured tokens inside and structured states of affairs outside.

Representation schemes have described the world. The point, however, is to change it. Let us center our representation scheme on metaphors of interaction. Representations play a mediating role in their owner's ongoing involvement with its surroundings. The point is not to *express* states of affairs but to *maintain causal relationships* with them. Instead of focusing on quasi-eternal statements involving proper names ("Snow is white," "Emus don't fly," "John loves Mary"), let's focus on statements about the state and progress of concrete activities ("The hammer I am using is too heavy," "This lane is being blocked by someone waiting to make a left turn," "The eggs I am beating are about as beaten as they're going to be").

Let us contrast two kinds of kinds of ontologies that a representation scheme might posit. *Objective* representation schemes posit a collection of individuals out in the world. These individuals are objective in that they exist independently of the existence or orientation or purposes of any observer. An agent refers to things in the world using a collection of constant symbols (and perhaps other, compound terms) in the represen-

tation scheme that correspond (potentially many-to-one since terms can codesignate) to some of the objective individuals in the world. First-order predicate calculus is an example of an objective representation scheme. (For an ambitious attempt at an alternative that nonetheless maintains the framework of model theory see Barwise and Perry 1983.)

By contrast, the a *deictic* representation scheme individuates things an agent's world *indexically*—in relation to the agent's body and identity—and *functionally*—in relation to the agent's ongoing goals and projects. Whereas one specifies the reference of an objective representation independently of its owner, the reference of a deictic representation depends critically on its owner's involvements in the world.

Although the utility of an indexical and functional ontology for a situated agent has no necessary connection with the metaphysical question of whether things in the world have an objective existence, the two issues are obviously related. The concept of objective existence, of course, has a long philosophical history and all the obvious formulations of it encounter grave difficulties. In *Being and Time* (1927), Heidegger presents a conception of ontology that transcends the metaphysical opposition between subjectivity and objectivity. Heidegger approaches the question not from the possibility of objective knowledge in science or religion but rather from the phenomenology of everyday activities. His analysis (in Chapter 3) of *worldhood*, in particular the notions of *significance* and *assignment*, influenced the notion of deictic representation.

The particular deictic representation scheme I will present posits *entities* in the world, such as *the-cup-I-am-drinking-from*, and *aspects* of these entities, such as *the-cup-I-am-drinking-from-is-empty*. The names of entities look like hyphenated noun phrases and the names of aspects look like hyphenated sentences that mention one or more entities. Each entity is specified indexically ("I") and functionally ("drinking") and mentions some role that a person, place, or thing can play in that activity ("cup"). A phrase like *the-cup-I-am-drinking-from* is *not* the entity itself, it is only a *name* for the entity. These names are only theorist's conveniences, not symbols in agents' heads. (Thus one should not be misled by the surface similarity between the names of entities and what Russell and subsequent philosophers have called definite descriptions. An entity is not at all a linguistic phenomenon. One should also resist the temptation to understand deictic representation in terms of Russell's distinction between definite and demonstrative descriptions, since this can only lead to confusion.)

Deictic representation schemes offer a very different account of abstraction from objective representation schemes. An objective representation scheme puts names, such as CUP-37, to each of the individuals involved in any episode. In order to represent knowledge that is independent of any particular situation one must put in a variable for each constant, perhaps expressing the abstraction as a universal quantification. Chapter C2 has already considered and rejected this approach. A deictic representation scheme does not make a distinction between the specific and the general. As a result, deictic representation transfers to new situations *passively*. If you know something about what to do when *the-cup-I-am-drinking-from-is-empty*, it will apply to whatever cup you

happen to be drinking from. If Monday's CUP-37 and Tuesday's CUP-38 have no salient differences, then the representation scheme will not distinguish them. The agent simply won't be able to tell them apart. Smith (1986a), following Perry, has spoken of passive abstraction in terms of the "efficiency" of indexical representation. (Similar ideas appear in (Evans 1982). Although Evans worked in the tradition of Frege and Russell, his treatment of indexical phenomena is remarkably subtle. For a broader, and I believe more useful, conception of indexicality based on careful study of naturally occurring activities see (Heritage 1984 Chapter 4).)

As a matter of vocabulary, if a particular agent is opening a certain door, the entity *the-door-I-am-opening* will be *bound* to that door; the door will be *assigned* to the entity; furthermore the agent, or perhaps some element of its machinery, will *register* the aspect *the-door-I-am-opening-won't-budge* according to whether the door is budging. It is possible for a given object to be assigned to two different entities at once. Here is an example:

> I was sitting at an outdoor café table drinking coffee and reading the newspaper. A light breeze arose and my paper turned up a corner and threatened to sail away, so I looked around for an object to use as a paperweight. The coffee mug suggested itself so I placed it on the newspaper. A little later I wanted to take a sip of coffee so I lifted the coffee mug. A breeze happened to be passing through just then, so the newspapers started sailing away again.

Mistakes like this one are consequences of assigning the same object two multiple entities. They are not inevitable consequences, though they would seem to be the default. Sometimes, perhaps as a result of an experience like this, one will begin to register an aspect such as *the-cup-I-am-drinking-from-is-the-same-as-the-paperweight-holding-down-my-papers* which will help one coordinate the two causal relationships or else separate them by selecting a different paperweight. A second example is equally telling:

> A friend of mine was working at a small software company on a project with a tight deadline. During this time he maintained a 120 hour per week schedule by snorting cocaine. Cocaine snorting tends to dry out the mucous membranes in one's nose. Because of this, cocaine users have a custom of snorting drops of water from their fingertips to alleviate the discomfort. So my friend had two different occasions to go to the water cooler, thirstiness or a dry nose. One day the water cooler was moved to a new location. He had no problem remembering the change when he was thirsty, but it took him a couple of weeks to break the habit of going to the old location when his nose was dry.

I am pleased to report that my friend has stopped doing this.

Because entities and aspects relate to the agent's material surroundings, it is important that they be operational. Being able to *express* "the cup I am drinking from is

empty" is all very well, but to actually get any work done one must *causally relate* the representational token to the physical reality. If the agent is to register the aspect *the-cup-I-am-drinking-from-is-empty*, it must, at the appropriate times, *track* the emptiness of its cup. Consequently, instead of speaking of the 'meaning' or 'semantics' of entities and aspects, we speak of their *causal relationships* to the material circumstances they mention.

The causal relationship between me and *the-cup-I-am-drinking-from* is a fact about me, about my habitual ways of interacting with the cup, about the cup itself, about my surroundings more broadly, and about physics. As I sit writing, my causal relationship with *the-cup-I-am-drinking-from* consists in my participating in a certain routine: keeping the cup on its coaster a little to my left on the table in front of me, picking it up repeatedly to drink and then putting it back down in its place, putting my eyes on it while going to grab it or put it down, and so forth. More broadly, these routines carry the cup to particular locations for making more tea, washing the cup out in the kitchen, letting it sit upright on its coaster overnight or else inverted on a towel to dry out after having been washed, and so forth. This is not to say that I would fail to recognize my cup were it to be randomly displaced by someone else; I am perfectly capable of recognizing it as my cup from its color and shape. These practices do not *define* the cup; they just constitute my causal relationship to it *qua the-cup-I-am-drinking-from.*

Establishing and maintaining my causal relationship to *the-cup-I-am-drinking-from* can take an arbitrary amount of work. At a formal dinner table it can require some reasoning and negotiation to assign water glasses to diners. At a bar, I have to try to keep track of whose glass was whose after buying another round. Whatever the patterns of interaction by which I keep myself in relation to this cup, they pick it out as being *the-cup-I-am-drinking-from.*

Having a causal relationship to an entity isn't a matter of extracting information from a single sensory impression but of maintaining an continuing connection between oneself and the equipment and materials of one's ongoing projects. As a technical matter, we need to relate the dynamics of these causal relationships to details of the agent's body, sensory apparatus, and ways of using this apparatus.

Think of all the entities involved in some activity as collected into a frame, in something like Minsky's sense of the term (1986, Chapter 24). Consider a particular activity, my custom of drinking tea while I work at the terminal in my office. As I work, I am engaged in a system of causal relationships with *the-tea-I-am-drinking* (I mean the actual liquid, which is currently Irish Breakfast, though when I don't need the caffeine so badly it's usually herb tea), *the-cup-I-am-drinking-from* (one of the three identical green coffee mugs I keep in my office), *the-handle-of-the-cup-I-am-drinking-from* (now facing off to the left), *the-coaster-on-which-I-rest-my-cup* (which I swiped from an English pub), *the-temperature-of-the-tea-I-am-drinking* (tepid), *the-tea-in-the-sip-I-am-taking* (sometimes this refers to a particular bit of tea and sometimes it doesn't), *the-hand-with-which-I-lift-the-cup-I-am-drinking-from* (currently my left hand). Each of these entities specifies a role in the ongoing routine of my drinking tea while sitting at the terminal in my

office. I take this routine with me when I work in other offices.

Following Minsky's original example (1974), one could equally well define the entities involved in a birthday party one is attending, *the-birthday-person-at-the-birthday-party-I-am-attending*, *the-presents-received-at-the-birthday-party-I-am-attending*, *the-birthday-cake-at-the-birthday-party-I-am-attending*, and so forth. It is important that each of roles in this enumeration is defined indexically and functionally, not objectively. It's not just any old birthday party, it's *the-birthday-party-I-am-attending*. A frame, on my deictic account, is different from a 'record structure'. A system using record structures could represent arbitrarily many birthday parties by making many instantiations of its generic birthday-party record. These records would indicate nothing about the agent's relationship to each party—I am attending it now, it is my own next birthday party, it was the best birthday party I ever attended, or whatever. Thus the records would be objective, not deictic. The kind of 'frame' implemented by FRL (Roberts and Goldstein 1977) is also a record structure, not a dynamic phenomenon like deictic representation. When Hayes (1977) claimed that frames were best understood in terms of the semantics of an objective logic such as an extended fopc, he was probably right about FRL-style frames, but not about deictic frames.

Ultimately, I would also like to understand the dynamics according to which the common forms of causal relationship arise and evolve. Deictic representation interacts with dependency maintenance in useful ways. Since reasoning in a deictic representation does not distinguish between situations that share the salient indexical and functional aspects, the dependency network will transfer a line of reasoning that has been saved in one situation to any other situation that shares those features, even if it occurs in a different setting and involves different individuals. I conjecture that most, if not all, transfer dynamics can be explained in this way. In particular, deictic representation is an important element of the dependency model of routine evolution discussed in Section B2e.

The particular deictic representation scheme I am describing is relatively simple. As the hyphens in the entities and aspects indicate, this scheme, by contrast to the usual practice in AI, does not posit any internal structure in its representational tokens. For example, nothing about the aspect *the-cup-I-am-drinking-from-is-empty* in itself indicates that it concerns the same entity as *the-cup-I-am-drinking-from-is-within-reach*. Other more sophisticated proposals might wish to relax this restriction if doing so turns out to solve any problems. Also, the scheme I will present has no formal semantics. I am not against formal semantics, but any systematic semantical analysis of a deictic representation scheme would have to differ from most existing ideas about semantics in giving a central role to the agent who owns the representational tokens.

Deictic representation is not supposed to be a theory of structured symbolic representation. Chapter C2 has already discussed some of the difficulties with theories of symbolic representation based on datastructures and has suggested investigating other forms of symbolic representation such as time-extended internal language or external written representations such as directions and recipes. Many forms of activity will pre-

sumably require agents to use internal language in addition to deictic representations. See (Chapman and Agre 1986, Chapman forthcoming) for the beginnings of an account of the relationship between the two forms of representation.

Deictic representation does not solve all representational problems. Regardless of the theory of representation one uses, one must choose particular categories to represent. The theory suggests that the categories must be operational (at least when they need to be) and reflect the actual indexical and functional significance of the entity for the agent, but any further guidance must come from an understanding of the dynamics of the domain. In practice, the decisions tend to be analogous to representational decisions in any other kind of representational scheme. For example, one must choose whether to represent *the-cup-I-am-drinking-from* or *the-container-I-am-ingesting-from* or *the-china-tea-cup-I-am-drinking-from* or some combination of them or something else entirely. A system that learns must be informed by some understanding of the dynamics influencing agents' choice of entities and aspect to represent. In building Pengi we made the choices ourselves, but the choosing was only difficult because understanding the detailed dynamics of the game was difficult.

Deictic representation is closely related to Gibson's notion of affordances (1979). Affordances concern the aspects of objects in the world that facilitate recurring forms of interaction between an agent and those objects. Flat horizontal objects afford putting things down, handles afford grasping, and so forth.

The theme of functionality also recalls Piaget's theme of egocentricity (see Singer and Revenson 1978 for a clear explanation). Piaget describes the emergence of the adult ability to treat objects as permanent and independent of one's own perspective. Piaget observes that this ability grows out of the child's experience with the invariant properties maintained by an object when the child interacts with it through multiple indexical and functional relationships. Some other related dynamic themes appear in Drescher's (1985) computational model of the stages of sensorimotor development describd by Piaget. He focuses on certain kinds of causal relationship that present the child with statistically measurable correlations that reflect regularities in the dynamics of the child's interactions with things in its environment. He also describes the concept of a *canonical perspective*, a dynamic effect that facilitates the discovery of deictic regularities by modifying the child's relationship to the world in recurring ways. An example would be the habit of bringing interesting objects into the middle of the visual field at a standard distance.

C3c Pengo and Pengi

Our domain is a reimplementation on the Lisp Machine of a commercial video game called Pengo. The player looks at a video monitor that portrays a number of discrete figures. The player has a joystick and a button that controls a cartoon penguin. The screen also contains cartoon ice cubes and bees. Pengo is a fairly difficult game. Playing it is an activity of moderate visual complexity and low motor complexity.

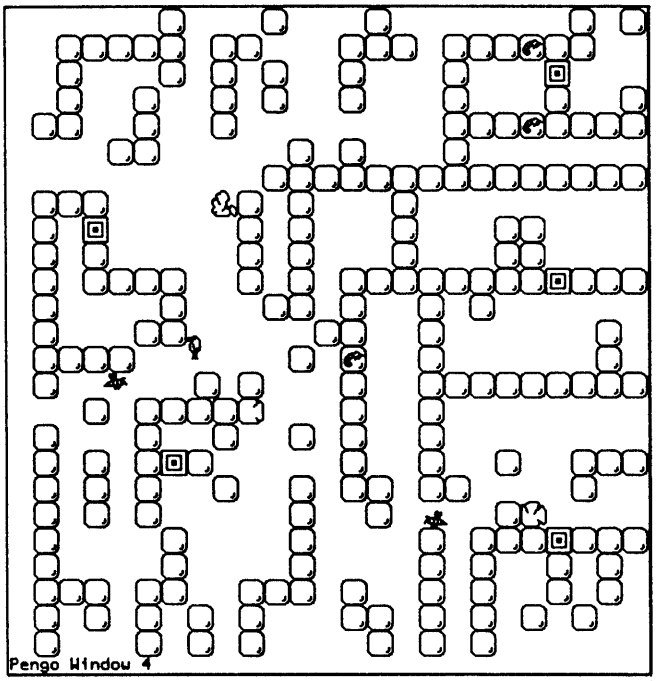

Figure C3.1. A Pengo game in progress.

The rules are as follows. The bees are trying to kill the penguin. If a bee gets too close to the penguin and stings it the penguin dies. If a bee or a penguin kicks an ice cube, the ice cube slides along in the direction it has been kicked. If it should happen to hit a bee or a penguin then that creature dies. Thus the bees can kill the penguin in two ways, by stinging it or by kicking ice cubes at it. The penguin can kill the bees in one way, namely by kicking ice cubes at them. The penguin has as its advantage is that it is presumably more intelligent than the bees, which operate by a very simple Markov process. The bees are always moving at the same speed. They tend to move in the general direction of the penguin; they randomly change their headings roughly every few seconds. If they find themselves able to kick an ice cube at the penguin they do so, but they don't go out of their way to position themselves behind an appropriate ice cube. The point is not at all to emphasize the element of opposition or antagonism between malevolent equals. We haven't made the bees particularly smart for the simple reason that intelligent, resourceful assassins are not a normal part of everyday life. Detailed knowledge about how the bees operate goes into an expert's playing of the game.

As a domain, Pengi is a big improvement on blocks world. Things move, the geometry is more complicated, the way objects are arranged in space is more meaningful, and the individual tasks relate in a clear way to an overall goal (winning the game). But at the same time, Pengi certainly does not capture many elements of the dynamic structure of everyday activity. Your breakfast is not out to kill you. Nonetheless,

the combination of goal-directedness and improvisation involved in playing the game is a fundamental and universal aspect of routine activity for which Pengi does offer a computational model.

Playing Pengo is certainly not a matter of executing plans. If you make a plan to sneak around behind a certain ice cube and kick it at a certain bee then that plan is usually not going to work. In fact, rigidly following such a plan is a good way to lose the game. Other bees will fly into range, the bee you are attacking will fly away, or the ice cube you have decided to sneak around behind will itself get kicked out of the way. Things go wrong in these ways on a regular basis. At the same time, playing Pengo is not a matter of continually reacting to things that go wrong. You need some notion of what you are trying to accomplish. You might want to get around behind some ice cube, or to run away down some channel, or to kick some ice cube out of the way. Your activity must be organized toward goals, but you have to stay on your toes.

Pengi plays a pretty decent game of Pengo. In its present state it is a little better than I am, which is to say that it wins from time to time and usually puts up a good fight. It ought to be able to support all the Pengo-playing dynamics we feel we understand. Chapman, though, who is extremely expert at the game, engages in some forms of interaction with the game that are more subtle than we've been able to describe, so perhaps the system cannot be made to exhibit some of the very advanced dynamics.

The player has a periphery and a central system. The periphery is fixed innate domain-independent machinery for early vision and low-level motor control. The central system is responsible for anything you'd call reasoning or making decisions. The interesting aspect of the architecture is that, even though Pengi does things for which conventional Planners would use enormously complex machinery, the central system is made entirely of combinational logic: gates and wires. It has no flip-flops, much less software. It has no pattern-matcher, no dynamic storage allocation, and no pointers. It has a clock, but the central system has no state.

Pengi constantly interacts with the game simulation. It is constantly generating actions, in the form of three bits representing joystick positions and button states. Our modeling of motor control is trivial since elaborate motor control is not a principal feature of this domain. Our modeling of vision, on the other hand, is fairly sophisticated. It matters how the visual system works.

The principal purpose of this chapter is to explain why combinational logic should suffice to be the central system of such a creature. The answer is going to be an instance of my thesis that understanding of dynamics leads to simple machinery. The machinery can be simple because we understand something about the dynamics of playing Pengo. Part of this understanding gets at the nature of improvisation and is fundamental to all routine activity. That is not to say that this architecture provides a complete model of making breakfast, or that it is an ultimate account of cognition. Certainly not, since there are interactional phenomena that arise in the world but don't arise in playing this game. Pengi, for example, can always see the whole game board, so it never has its back turned on anything. Nonetheless, that Pengi's central system can be made wholly

of combinational logic is a substantial claim.

Briefly, Pengi's central system can be made of combinational logic for two reasons. One reason is that Pengi embodies an account of representation, namely deictic representation, that can be readily implemented with combinational logic because it does not involve variables and pattern matching. The other reason is that Pengi can engage in an orderly, flexible, goal-directed interaction with its world without maintaining a plan or world model, so the central system does not need to keep any state. The next few sections explain these points in more detail.

C3d Entities and aspects in Pengi

On just about any conventional account, an agent that played Pengo would maintain a world model of the evolving game board. This model would assign names such as BEE-34 and BEE-35 and ICE-CUBE-61 and ICE-CUBE-62 to all the individuals in the world, or at least the ones that matter. Likewise, the world model would contain symbolic representations of all these individuals' important properties and relations. The world model would be 'maintained' by a process, presumably involving perception, that kept track of changes in the world and updated the world model to reflect them. (Most Planning systems assume that the world model is maintained automatically, but see (Chatila and Laumond 1985) for an impressive study in the complexities of actually trying to maintain an accurate world model in a grossly simplified environment.) Decisions about what to do next would involve inference from that world model to formulate plausible courses of action and simulate their likely effects. This is an objective style of representation insofar as its elements correspond, without regard to the agent's location or heading or projects or attitudes, to objectively individuated objects in the world. This conception of representation has already come in for some abuse in earlier chapters.

Deictic representation is ideally suited to an activity such as playing Pengo. These are some of the entities that Pengi has to keep track of at various times.

- *the-ice-cube-I-am-kicking.* This is simple enough. The "I" really refers metonymically to the penguin; a more accurate name would be *the-ice-cube-which-the-penguin-I-am-controlling-is-kicking.*

- *the-direction-I-am-headed-in.* Entities do not have refer to concrete objects. They can refer to directions, in this case up, down, left, or right. This is not to say that Pengi has a variable called *the-direction-I-am-headed-in*, only that Pengi is in a certain causal relationship to the direction in which it is heading.

- *the-bee-I-am-attacking.* If Pengi is attacking some bee over some period, it might dance around and try to get behind various ice cubes to kick at the bee. The whole while, in virtue of being in that pattern of interaction, that bee will be *the-bee-I-am-attacking.*

- *the-bee-on-the-other-side-of-this-ice-cube-next-to-me* is good to keep an eye on. If the penguin is aligned with an ice cube and a bee, it is important to keep an eye on the bee since it could kick the ice cube, and if it can't then maybe Pengi can kick the ice cube at it.

In general, Pengi's entities will refer to the objects near the penguin. In this sense, Pengi has a strong sense of focus. Objects and events off at the other end of the board probably won't have any functional significance for the player, so they usually won't be the referent of any entities at all.

Pengi's central system can get along without a pattern matcher or other complex variable-binding facilities because deictic representation permits it to generalize across indexically and functionally equivalent situations by simply not introducing spurious distinctions among them. If Pengi attacks BEE-34 on one moment and BEE-35 the next, an objective representation scheme would force those two episodes to be represented differently. But all that matters to Pengi is that it is engaged in a certain routine pattern of interaction with *the-bee-I-am-attacking*. If Pengi knows what to do about *the-bee-I-am-attacking*, it will do it in each case without bothering to distinguish. They are functionally identical cases.

As Pengi actually decides what to do, it reasons in terms of *aspects* of the various entities. If an entity sounds like a noun phrase, as in *the-bee-I-am-attacking*, then an aspect will sound like a sentence, as in *the-bee-I-am-attacking-is-running-away-from-me*. It is an urgent situation when *the-bee-on-the-other-side-of-this-ice-cube-next-to-me-is-closer-to-the-ice-cube-than-I-am* since the bee could run up to the ice cube and kick it at the penguin before the penguin could run up to the ice cube and kick it at the bee. Consequently, Pengi had better get the penguin out of the way. The situation is less urgent if *the-bee-on-the-other-side-of-this-ice-cube-next-to-me-is-moving-away-from-the-ice-cube*, though. These are a useful pieces of knowledge about the game, but in themselves they don't explain how they are operationalized and how Pengi actually carries them out. That is the next section's topic.

C3e How Pengi decides what to do

Figure C3.2 is a diagram of Pengi's best trick. In the diagram is a bee that isn't lined up with any ice cube. Lacking any way to kill the bee, Pengi is reduced to waiting around for the bee to align itself with an appropriate ice cube. An alternative, though, is for Pengi to move an ice cube so that it aligns with the bee. You can't win the game without doing this regularly. In the example in the diagram, Pengi can kick the ice cube on the left so that it hits the ice cube on the right. (Momentum is not conserved in this game. The moving ice cube stops dead when it strikes the stationary one.) Now that the ice cube is aligned with the bee, the penguin can move around behind it and kick it at the bee.

One might be tempted to refer to this trick as a plan. Yet Pengi neither makes nor uses plans. The two-ice-cube trick is one of the routines we foresaw when we

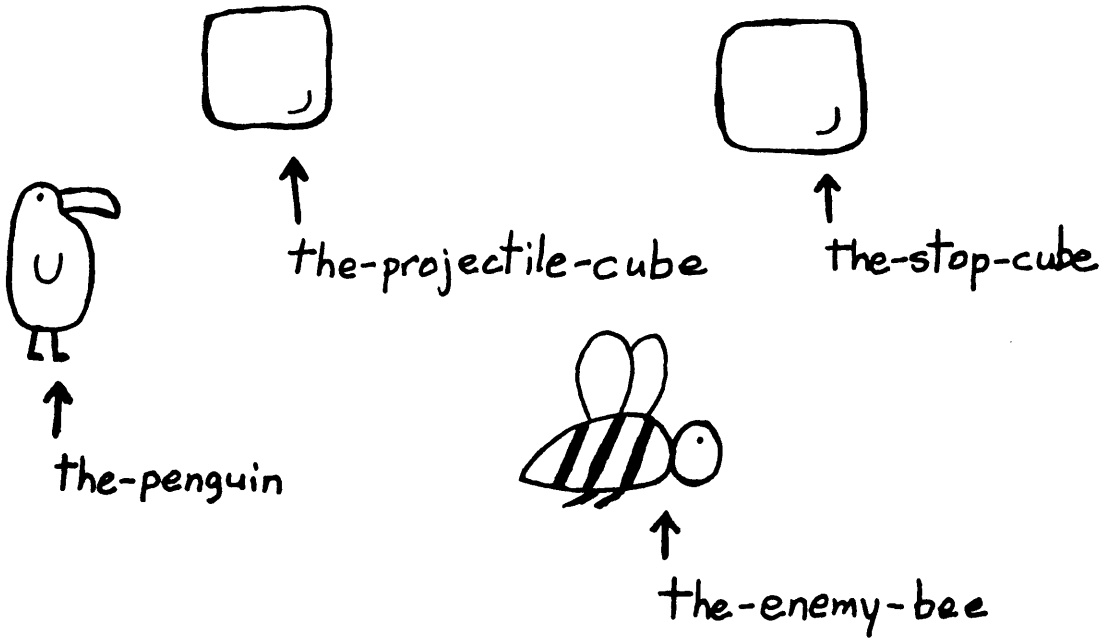

Figure C3.2. Pengi's best trick is to move one ice cube into position by kicking it up against another one.

built Pengi's central system circuitry. Pengi engages in the routine because it knows what to do in each situation that arises in it, not because it refers to a plan or other representation of the routine. Let us contrast the way Pengi performs this trick with the way a conventional Planner might perform it, on three counts.

One count is, why would each kick the ice cube in the first place? How does it know that this is a good move? A traditional Planner would do a simulation in its world model and, through inference over some logical codification of the geometry, deduce that this ice cube can be gotten from there to there to there. Pengi does something different. Instead of performing inference against a world model it *visualizes* against the world as it actually sees it. The visual system is good for visualizing things against the world because it provides a set of *visual operations* that have functional significance in many domains: coloring in regions, finding out whether things are lined up, finding something of a given shape or color or style of movement in a given vicinity, telling whether anything is located between two designated points, keeping a finger someplace for future reference, and several others. These visual operations are combined into *visual routines*, by means of which the central system uses the visual system to register the current values of various aspects of its environment. The actual situation will contain all kinds of visual clutter not shown in the figure, but by running these visual operations Pengi will be able to visualize that these two channels are clear and that the second ice cube is in position to stop the first one.

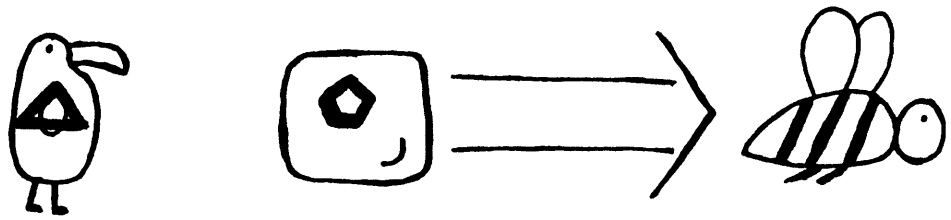

Figure C3.3. Pengi visualizes where the ice cube the penguin has just kicked will end up by using its visual operators.

We wrote Pengi's visual routines ourselves. But a growing expertise at any activity, from cooking to mathematics, includes acquiring a set of visual routines and learning to use the visual operators more efficiently, more sparingly, and more accurately, choosing the right one for the job. Beginners use their visual operators in fairly crude, generic, safe, heavily redundant ways; as they become skilled their visual routines evolve just as all routines evolve. Future work needs to elucidate the dynamics by which visual routines evolve in the context of larger activities.

Chapter C4 will explain Pengi's visual operators and the technical issues around them in detail. For the moment let us consider an example. Figure C3.3 shows how Pengi finds *the-ice-cube-that-the-ice-cube-I-just-kicked-will-collide-with.* Pengi has just kicked this ice cube and has a marker tracking it. It also knows the direction the ice cube is moving. So it uses an operator that starts on the marker, moves to the right, and drops a second marker on the first thing it hits. Once this operation completes, Pengi can use further visual operations to register various aspects of *the-ice-cube-that-the-ice-cube-I-just-kicked-will-collide-with.*

The second count on which we might contrast Pengi with a conventional Planner concerns why each would kick the ice cube the *second* time. A conventional planner would take that step because the program counter in its executive is equal to two. The Planner's executive chooses the next Plan step to execute by following a pointer to some structure like `(kick a)`.

Pengi, by contrast, kicks *the-projectile-cube* because it is lined up with the bee. It sees a clear channel, an ice cube, a bee, and a penguin in the right spatial relationships, so it moves the penguin behind the ice cube and kicks it at the bee. That the ice cube got there by having been kicked by the penguin earlier on doesn't really affect why it is a good reason to kick it at the bee. Pengi is, on each next moment, deciding what is the right thing to do in the world as it is then.

Pengi's ability to rederive its course of action from the evolving game situation makes it more flexible than the Planner in the face of unexpected situations. The bee might fly out of range, or another bee might fly into range to make it dangerous to traverse this territory here. Or some bee might kick *the-stop-cube* out of the way. Pengi chooses the best actions to take in the world as it finds it right now. That one's program counter equals two is not a good reason to do anything. A program counter's connection to the actual situation outside is far too tenuous to be an agent's principal resource in choosing actions.

Pengi's policy of continually redeciding what to do is also an improvement on the conventional technique of execution monitoring. An executive might be asked to monitor all the conditions that would indicate that things are not going according to Plan. The original and simplest version of this proposal is to associate with each Plan step a set of "preconditions" that the executive can check on before executing that step. Even assuming that provision has been made for cases where the monitored conditions aren't operational or cannot be checked without potentially complex manipulations, monitoring can only determine when a given Plan is going wrong, not when a different Plan would be an improvement. Pengi regularly aborts one of its routines when a better opportunity arises. Having pursued the opportunity, it might return to the original routine if that seems like its best option when the time comes.

The third count is machinery parsimony. Both Pengi and the conventional Planner can perform the two-block trick; which one does so with the simpler machinery? Clearly Pengi's is simpler. Pengi has no need of structural Plans (either upper- or lower-case) or pointers. All Pengi needs is the ability to look at the world and decide what to do, something any situated agent needs. Pengi has just enough state to keep a finger on the important aspects of the world, but it doesn't have a whole full-blown model of the world in its head. An understanding of dynamics leads to simpler machinery.

In summary, Pengi's scheme of continually redeciding what to do based on what it observes in the current situation is both simpler and more flexible than the conventional Planner's scheme of making a Plan and executing it until something goes wrong.

It helps to think of Pengi's strategy as a dynamic version of Newell, Shaw, and Simon's technique of difference reduction (1960). Difference reduction was originally a technique for searching problem spaces. Given a state known to be reachable from the initial state, the problem solver computes a set of *differences* between that state and the goal state. It then indexes those differences into a table to discover which ones it can reduce by applying an operator. The operator will produce a new state. If that new state is the goal state then, in Newell, Shaw, and Simon's vocabulary, the problem

is considered solved. If not, difference reduction can be applied to the new state. This method can be repeated until the problem is solved. None of this happens in the real world. To 'apply' an 'operator' does not involve taking an actions in the world, only simulating the action's effect.

Dynamic difference reduction, by contrast, happens in the real world. The agent alternates between finding something to do and doing it. In the case of the two-ice-cube trick, each of the player's actions reduces a difference: kicking the ice cube once gets an ice cube lined up with a bee and kicking it again kills a bee. One could imagine an agent making an omelette the same way, repeatedly looking for things that need doing and then doing them. Using this method requires some knowledge of what steps will need to be taken, but it requires no explicit representation of serial order. The agent might take the actions in different orders on different occasions, according to the happenstances of noticing. But so long as it understands the preconditions of its actions—almost all of which are readily visible and furthermore enforce themselves by rendering the actions impossible by their absence—it will get the job done each time. Since these actions take place in the real world and not in simulation, of course, they cannot be retracted by simply moving to a different location in the search space. It helps if an agent engaging in dynamic difference reduction knows about difference reductions that lead to wasted physical effort, just as it helps if a search method knows about paths that lead to wasted search effort.

Pengi certainly has limitations. It is hard to be precise about where the capabilities of Pengi's architecture leave off—it could easily be the case that such machinery can participate in dynamics we haven't observed or understood yet. Nonetheless, Pengi was only designed to participate in a certain collection of dynamics, not in the whole of human activity. Other domains might require Pengi to have other capabilities. For example, if Pengo got harder, Pengi might sometimes have to refer to a plan. The plan would explain how to deal with some tricky situation, or perhaps what strategic issues bear on the matter of which bees to attack when. The plan wouldn't be exhaustive like a program because Pengi isn't dumb like a programming language interpreter. Instead, the plan might consist of natural language, or something very much like it. Other activities will require a central system to maintain state, to use its visualization abilities without any domain materials being present, and so forth. Precise specification of this additional machinery should await an elucidation of the relevant dynamics.

C3f Seriality and focus

Pengi can have such a simple architecture because we went to the trouble of understanding something about the dynamics of situated activity in general and Pengo-playing in particular. Central to this understanding is the fact that a situated agent, far from being a detached abstract intelligence, has a body. Having a body implies a great deal: facing in a direction, having a location, the tendency of the objects near us to be the significant ones, and more generally all the resources the physical world provides to aid

us in our activities.

This section concentrates on two intertwined themes that figure in the dynamics of any embodied agent's interactions with its environment: seriality and focus. These themes of seriality and focus manifest themselves on three different levels.

The first level is in the realm of dynamics. Simply having a body imposes a certain seriality on your activity. You can only be in one place at a time. You can only be facing and looking in one direction at a time. You can only do about one thing with your hands at a time. You can only drink from one cup at a time. If you have several things to do, you have to do them in some serial order.

The second level is that of the visual system. The themes of seriality and focus assert themselves again. The visual system, as we've seen, employs visual markers, in terms of which many of the visual operations are defined. Recall the parking-lot example from Section A3e—if I am using my visual system to map out the free space I am going to walk through when I am walking through a parking lot, I can't be using that operation to mark out free space for some other purpose as well. I am focused on one thing at a time in the sense of my eyes being aimed in one general direction at a time, in the sense of foveating in a particular place, and in the sense of my visual markers being allocated to certain locations and not others.

Finally, the themes of seriality and focus assert themselves on the level of representation. I can only have one *the-cup-I-am-drinking-from* at a time. I am only going to be involved with one such cup at a time. There is a strong claim here, that if I actually want to deal with several such objects it has to be one at a time.

All these kinds of seriality and focus, as it were, line up. The representation scheme is less general than others in that it can't represent arbitrary spaces of meaninglessly distinguished objects with identical significances all at once. But then it don't need to, since an embodied agent can only interact with a small number of functionally distinguished objects at a time.

In the end, this is the main way in which our understanding of dynamics has led us to simpler machinery. The machinery only provides the capacities we really need and takes advantage of the inherent properties of interaction in relatively benign worlds for its power rather than relying on complete generality.

Despite these lessons, Pengi only has a body in a limited sense. It stands transfixed in front of its video screen without moving among the materials and equipment of its task. Nothing is ever behind it. It interacts continually with the world but it is not truly *in* the world. More genuinely autonomous embodied agents will require a deeper understanding of the dynamics of embodiment.

It would be instructive to try applying Pengi's technology in a different domain. Doing so will presumably reveal that we made many mistaken decisions about this architecture's underconstrained features. Some of our mistakes will have resulted from the exigencies of having to make it programmable at all, given our rough understanding of the detailed dynamics of this particular activity. Other mistakes will be due to the the kinds of biases this domain has forced on us. A Pengo board is a very visually busy

place. Since the bees have a uncooperative random element, a Pengo player must keep a finger on an abnormally large number of details of its environment. That things are clustering about you, that there are emergencies, that you need a fairly diffuse kind of focus, are different from making breakfast, where the simple fact of your having a body determines a fairly tight focus.

C3g Objections

Pengi reliably provokes a long list of questions. Though I have tried to deal with most of these while developing the argument, several are best dealt with here in isolation. I have tried to word most of these questions in exactly the way people tend to ask them.

> You say that Pengi doesn't use plans. But couldn't we view its combinational logic network as a plan?

I suppose we could, if we used the word 'plan' so broadly as to border on uselessness. But the vocabulary of plans and planning is hardly the most useful way to look at Pengi-like networks. The plan language in question would not look much like any existing programming language or the plan language of any existing Planner. It seems very unlikely that any planner could understand enough about the dynamics of activities such as Pengo-playing to generate networks such as Pengi's in the fashion of conventional Planners.

> You say that Pengi isn't a Planner. But how can it be a serious model of any kind of activity without looking ahead into the future?

A system can look ahead into the future without being a Planner. All that is required is that one's actions take sufficiently reliable ideas about the future into account. In a Planner, this takes two forms: the decompositions of goals into subgoals provided by the programmer and the simulations of possible future courses of events that the Planner performs. Pengi also looks ahead into the future, in two senses. First, Pengi's tactics were designed using an understanding of the dynamics of the game. This understanding involves ideas about what tends to happen next and about how individual actions lead to winning the game. Second, Pengi uses its visual system to visualize possible courses of events. Chapter C4 will demonstrate Pengi's visualization in more detail. Pengi does not have a general facility for simulating the future. It doesn't need one.

> You say that Pengi doesn't use world models. But couldn't we view its retinal image of the Pengo screen as a world model and its visualizing as simulation?

The distinction between performing a simulation and running visual routines is indeed not as clear-cut as it appears. Perverse as it would strike me, one might choose

to view the visual operations as performing some sort of logical deduction. Visual routines certainly do not constitute a general simulation or theorem proving facility, but inference need not imply a general inference facility. The important difference is that the representations over which the visual operators apply are retinocentric and thus not objective world models. A retinal image is a poor world model because it only encodes the very most primitive information about intensities of light. It does not encode the identities or structural relations of the visible materials. Furthermore, a retinal image is not a world model because its relation to outside events depends on the agent's location and orientation. Pengi, it is true, always has the same location and orientation, but other systems employing the same technology need not.

> You say that Pengi doesn't use variables. But couldn't we view entities as variables? Couldn't we view the visual system's markers as variables?

I suppose we could, if we used the word 'variable' so broadly as to border on uselessness. We can use the word 'variable' in any fashion so long as some words are left over to express the intended distinctions. A variable, unlike either an entity or a visual marker, participates in a relationship between two abstractions inside a machine, namely the variable and its current value. Entities and markers, by contrast, participate in a relationship between an agent and a thing in the world, namely the object currently bound to the entity or the object the marker is currently tracking. In each case, the entity or marker, unlike a variable, participates in a *causal relationship* between an agent and certain materials in the world.

> You emphasize the way Pengi's entities support a sense of focus. Couldn't you implement focus in a much more straightforward way, perhaps by only maintaining a model of what's happening within a certain radius from the penguin?

That wouldn't do, since a bee can kick an ice cube from an arbitrary distance. It would also be a domain-specific solution that would not be very helpful in real-world domains where it would be prohibitively complicated to model all the materials within any reasonable radius of anything, be it the agent's gaze or hands or tools or whatever.

> You say that deictic representation does not involve constant symbols or proper names. This is convenient enough for Pengi given that it lives in a world where all the objects are perfectly generic. But what if Pengi needed to keep track of a particular bee, say because it was especially dangerous? If Pengi were dealing with real people rather than cartoon bees, wouldn't it want to have separate names for all of them?

This question is much more complex than correspondence theories of representation make out. Simply *expressing* an object's individuality is only the easy part of keeping track of it. Next time you play a video game, pick out an arbitrary monster and try

to keep track of it as you play the game. It is next to impossible. In reality, people can usually identify functionally distinct objects because indistinguishable things tend to be interchangeable. To keep track of an object whose special functional significance is not obvious, we must resort to such methods as marking it or putting it in a special place.

Since every person is interestingly different from every other, people make an excellent opposite extreme from cartoon bees. You only learn and remember a person's name, though, once they acquire a distinctive significance for you. When you sell people hamburgers at McDonald's or give directions to strangers, you rarely get any glimpse of their individuality and you hardly ever learn their names. Once you get to know someone well enough to have any use for their name, you are certainly best advised to remember basic facts about them, but do you really store these facts by predicating them on the person's name? When you run into a recent acquaintance on the street, you regularly remember how you met them and what they do even though you can't remember their name.

There can be no doubt that the ability to express unique identities, such as through proper names, is sometimes useful. But proper names are such a complex phenomenon that they should not be made into a primitive representational element. Proper names, like a great many other complex representational techniques, are part of natural language. Pengi doesn't use natural language. If Pengi *did* use natural language it would not *replace* deictic representation. Instead, the strings of natural language would be resources among others that the agent would use in getting things done. Deictic representation, not natural language, is the fundamental type of representation.

> You say that deictic representation is different from traditional logics such as fopc. But couldn't we use fopc to express reasoning using deictic representation by including an indexical constant term "I"?

Expressive power is over-rated. First-order logic can express all kinds of things, but it offers no account of epistemics. Deictic representation, by contrast, concerns the causal relationships that permit agents to keep track of objects. Simply expressing an idea in deictic terms isn't any help. The point is to describe the causal relationships into which the various entities and aspects enter. Simply positing a relationship between "I" and the agent doesn't address the problem.

> You criticize conventional representation schemes for having an objective, correspondence semantics. But this isn't how people in AI use the word 'representation'. A representation is just a structure that a program manipulates. Aren't you attacking a straw man here?

Let's get a few distinctions clear. I did not say that every representation scheme ever created has an objective semantics. The story structures that have been manipulated by many programs at Yale, for example, resemble natural language in some principled

respects. Many representation schemes have had no clear semantics at all, despite an evident desire to represent the outside world.

I refer to objective representation schemes as 'conventional' because correspondence semantics, usually in the form of a compositional semantics *a la* Tarski, is the only serious, worked-out theory of semantics, philosophical or mathematical, that is widely used in AI research. To the extent that most AI representation schemes can be said to have any clear semantics it is an objective semantics. Hayes' argument, in "In defense of logic" (1977) that practically all AI representation schemes boil down to first-order logic is over ten years old now, but so far as I am aware it is still accurate.

Within computer science, there is indeed a widespread practice of referring to all datastructures as 'representations'. I consider this usage a pretentious corruption of the word, but that's not the point. The point is that a great many of the datastructures employed by AI programmers are plainly intended to have some reference to actual, particular things in the outside world. They include the names of physical and social individuals, they mention physical and social processes, and so forth. The practice of building systems that construct and manipulate these structures is a sham unless there exists some account of the way in which they relate to the actual individuals and processes they mention. This is the job of a theory of representation.

In insisting upon some theory of representation, I do not mean to advocate the conventional explanations of representations in terms of meaning and reference, much less the representational theory of intentionality as a whole. But people use representations, in one way or another, and representations relate to the world, in one way or another. (For an enumeration of some other options, see Smith 1986.) Whether we want to do psychology or engineering, we need a theory of these phenomena.

> Isn't Pengi's central system just a finite state machine? And hasn't it been proven that there are very strong constraints on what a finite state machine can compute?

Technically speaking, Pengi's central system is not a finite state machine since none of its output wires re-enter immediately as inputs. All of the output wires lead into the visual system or the motor system, where they can cause various events whose effects might be detected in the next cycle's inputs. As for the FSM computability theorems, these theorems all concern the functions from inputs to outputs a given type of machine can or can't 'compute' in the sense of getting every case correct. I have no idea how to relate them to the dynamics of any particular real activity such as making breakfast, where false starts and cut corners are so common. Beyond all this, the critical point is that Pengi by itself doesn't really 'compute' anything at all. It is in Pengi's interactions with its world that work gets done.

> Pengi doesn't have any state in its central system, but surely people remember things.

People certainly do remember things. Pengi is not supposed to be a model of all human activity. Pengi does not have any state in its central system purely as a matter of machinery parsimony. Pengi's central system doesn't *need* any state because we studied the dynamics of the activity Pengi is intended to engage in. Other domains will presumably be different.

If I have a bias against state, it is a reaction to the extreme practice of building world models. A situated agent's environment provides vast resources for keeping track of its materials and activities. Why maintain a model when you can look and see? The world makes more sense in a glance than anything could prove it makes in an hour.

> Pengo is such a specialized domain. Why should we consider that Pengi has taught us anything about everyday life as a whole?

Pengi's ability to play Pengo certainly proves nothing about making breakfast. I don't plan to demonstrate any propositions about everyday life by generalizing from Pengi. Instead, Pengi is an illustration of some things I feel I learned about everyday life by moving back and forth between observation and pencil-and-paper technical exercises. I believe what I do about everyday life because I went and looked at it. Chapman and I chose the Pengo domain because we couldn't build a breakfast simulator that did justice to the reality of breakfast-making, because we feel we understand most of the steady-state dynamics of Pengo-playing, and because the central dynamic themes of improvisation, contingency, and goal-directedness all arise in Pengo-playing in natural ways. To be honest, the Pengo domain was Chapman's idea and it took me a while to reconcile myself to it. Its only drawback as a first experimental domain is its adversary nature.

> It seems as though playing Pengo involves reacting to a series of crises. Combinational logic is going to be very good at this, but what about anything else? Haven't you cooked the Pengo domain to show off the strengths of your architecture and hide its weaknesses?

Playing Pengo isn't like that at all. A Pengo player that does not use complex tactics and anticipate the future then you will indeed find itself reacting to a series of crises. Beginners tend to find themselves alternating between periods when they're attacking and periods when they have gotten in trouble and have to focus on surviving. More advanced players must often defend themselves, but the line between attack and defense is more blurred. Pengi itself is a relative beginner, but it spends at least as much time on the attack as it does dealing with crises. In particular, Pengi constantly uses its visualization abilities to see if it can use its various tactics.

> You insist that the world is fundamentally a benign place, yet your domain is violent. If the efficacy of improvisation depends on the beneficence of the world, why should we believe in your analysis of why Pengi can play Pengo?

When I worked at Atari we used to sit around and bemoan the seeming fact that all the simple, interesting, comprehensible things one can do in real time in simulation on a normal-sized computer involve violence. The adversary nature of Pengo is unfortunate, but we can be precise about the way in which it disrupts our efforts to build Pengi.

The principal difficulty is that Pengi, like any human player of Pengo, must continually work to keep track of the bees because of the random component to their motion. A Pengo-player cannot take advantage of many of the dynamics by which one can keep track of things in other domains, such as keeping them in a special place, carrying them in a pocket or purse, going and finding them when a need for them arises, or relying on them to send you an occasional letter. As a result, playing Pengo places much heavier demands on one's visual system than most normal activities. As Chapter C4 will explain at length, the most difficult thing about writing Pengi is to arbitrate among the visual operators that Pengi uses to keep track of all the bees and their geometrical relations to the penguin. That the only serious difficulty in writing Pengi corresponds so closely to the principal difficulty in playing Pengo is encouraging.

At the same time, the video-game metaphor of killing-or-being-killed shouldn't distract us from the benign properties that the Pengo domain *does* share with routine activities in the world of everyday life. The materials of the player's immediate activity are readily visible. The vast majority of its objects—the ice cubes—do *not* move about capriciously. The domain has a strong sense of locality since objects do not move quickly compared to the dimensions of the game board. And the player does not encounter wholly unfamiliar types of objects.

Furthermore, it is important to distinguish Pengo from activities that involve genuine violence. One simple distinction is that in Pengo it is the penguin that dies, not the player. But there is a deeper distinction as well. The world of Pengo works in a definite, unvarying way. The bees are always driven by the same simple Markov algorithm. Pengo would be much harder if the bees were smarter. The bees could, for example, observe patterns to the player's actions and devise strategies to take advantage of them. If Pengo's bees changed their flight patterns, Pengi would certainly compensate to a limited extent, but if the changes were substantial then Pengi would not be able to formulate the new representations that adaptive strategic changes would require. If someone is genuinely trying to kill you—if you are organizing a war or being hunted—then the way you represent the world is your Achilles' heel. All the unarticulated assumptions behind your reasoning and all the unreflective patterns to your actions create opportunities to catch you off guard. For example, as weapons systems and their logistical support grow more complex, and as the components that must work together grow more numerous, the opportunities for subversion and sabotage multiply. None of this is at issue in Pengo or other formalized wars such as chess. In such domains you can characterize the entire space of possibilities *a priori*. I don't advise constructing explicit representations of those spaces of possibilities, so long as your tactics implicitly take their properties sufficiently into account through your understanding of the dynamics of playing the game.

The lesson here is that part of the benign nature of both Pengo and the everyday world is that, far from working to subvert your understanding of them, both help ensure that the representations you've developed will continue to work well enough. Pengo offers this assurance through its simple lack of change. In the everyday world, though, the story is more complicated. The unvarying nature of physical laws and the tendency of physical objects to stay where you've put them certainly help your representations to keep on working. Beyond this, considerable effort goes into keeping the physical apparatus of everyday activities tidy and in working order. But above all, the order of the everyday world is a social order that is actively maintained through the intricately organized efforts of all its participants. Far from subverting the common reality, everyone makes everyone else participate in the common task of maintaining it (Heritage 1984 Chapter 4). The vast majority of this work is invisible to casual inspection, but careful investigation or experiments in deliberate disruption can easily reveal it. Every element of the everyday world retains its significance and its salient properties through this continual cooperative effort. For a striking examination of this proposition in the context of a particular door see (Latour 87).

> Does Pengi really understand what it is doing? After all, you built the circuitry in Pengi.

Lacking a satisfactory technical or philosophical interpretation of the word 'understanding', it is hard to speak plainly to this objection. The short answer is no, Pengi does not understand what it is doing. No computer has ever understood to any significant degree what it is doing.

We built Pengi's circuitry ourselves and the authors of Planners provide them with the possible decompositions of their possible goals and with functions for computing the consequences of their actions. What grounds could we have for ascribing some measure of understanding to a conventional Planner? One ground might be the amount of relevant material made explicit by the Planner's representations. Without computationally tractable procedures for manipulating these elaborately structured representations, though, why should we credit the Planner with understanding them to any greater degree than Pengi understands the mnemonic labels, such as *the-bee-on-the-other-side-of-this-ice-cube-next-to-me-is-moving-away-from-the-ice-cube*, that we place on its wires? Completely general systems are easy to define, but they don't properly exist until they really work.

Chapter C4

How Pengi works

C4a Context and summary

Recall that our project is to relate ideas about the organization of the everyday ordinary routine activity of ordinary people to computational issues of architectures and circuitry. The challenge of the project lies in the large range of issues involved, all the way from whole ways of life down to details of circuitry. The last chapter was somewhere in the middle of that range. It concerned the elements of everyday activity that I have broken off and made my starting place, an architecture that fits with those ideas about activity, and an overview of a system Chapman and I have built that embodies that architecture.

This chapter discusses the system's technical details—what the wires mean, where the clock lines and delay lines are, and so forth. I will try to relate these small details of the machinery to larger dynamic issues, if not to making breakfast at least to the larger organization of playing this particular video game.

Section C4b describes the architecture as a whole and addresses the computational significance of the distinctions between its two principal components, the visual system and the central system, and the well-defined interface that joins them.

Section C4c discusses the visual system. It presents a list of the particular visual operations implemented by Pengi's visual system and a set of criteria for evaluating this list.

Section C4d discusses the central system. The central system is made of combinational logic whose construction requires some sort of principled methods.

Section C4e puts these elements together and goes through an extended example showing how the penguin kills a bee. Many issues arise, such as the arbitration of conflicting claims on visual operators and the connections of this particular scenario to the larger dynamics of the game, especially the pervasive consequences of contingency.

C4b Architecture

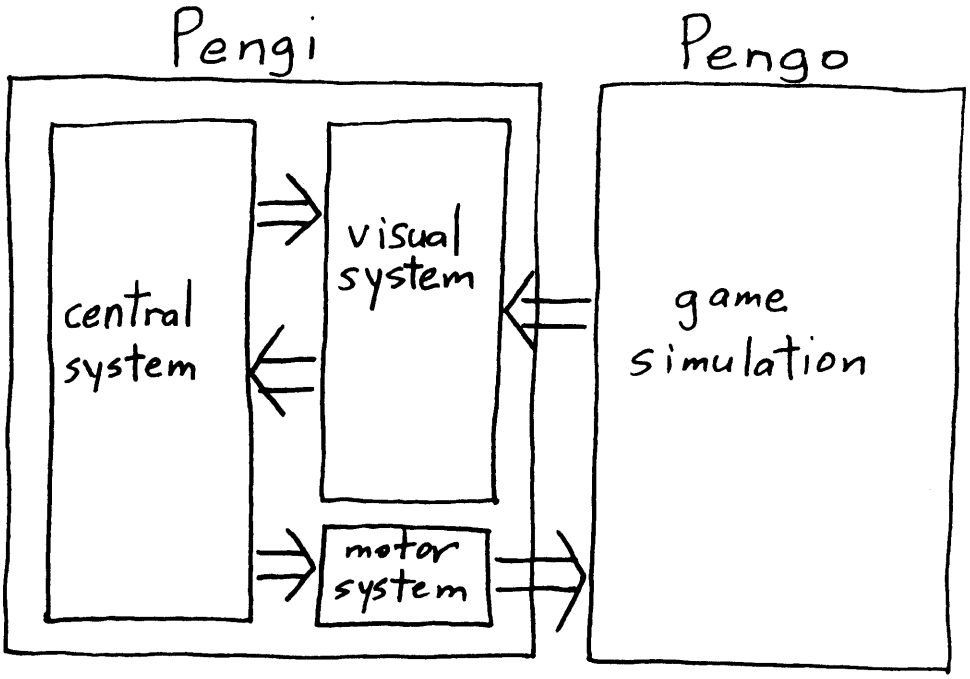

Figure C4.1. Pengi is composed of a central system made of combinational logic, a moderately realistic visual system, and a trivial motor system. It interacts with its simulated world on a fairly fast clock.

Figure C4.1 shows the top-level modularity of the system. We need to consider the various boxes and arrows on two levels. One level is the ideal of an actual video game from whose video-out an actual cable leads into the frame buffer on a Connection Machine, which performs early visual processing and connects to the parallel central system hardware. In reality, though, large parts of this are simulated. The game runs on a Lisp machine. There is no early visual processing because we don't have a sufficiently robust and integrated set of vision subroutines for a Connection Machine. Instead, the visual system simulates its early vision by reaching into the game's internals and figuring out what the outcomes of various visual operators would be. The simulated visual system certainly sweeps some important issues under the rug. The operation of the central system's circuitry is simulated as well, but since the simulation is down at the wire-and-gate level fewer important issues are elided there. Since our principal concern here is not early visual processing, and to suppress unnecessary implementation detail, I am going to proceed as if the system were organized according to the ideal.

The vision and central systems together form the player and they close a loop with the game itself. Indeed they close a very tight loop; the game moves fast and the player must act continually. The wires on the motor control bus continually need new values.

Implicit in Figure C4.1 are several assertions about the architecture. One claim concerns the boundary between the visual system and central system. The architecture offers an answer to the longstanding question concerning the relationship between a visual system of the sort suggested by Marr—innate, bottom-up, domain-independent, massively parallel, probably analog—and the central system—which isn't wholly innate or so easily partitionable into modules. I believe in a sharp distinction between peripheral systems and a central system. (I say 'central system' in singular since I don't think of it as modular.)

It does not matter how modular the peripheral systems are, so far as this architecture is concerned, provided they are distinct from the central system. Nor is it wholly certain where the peripheral-central boundary is: how far along that bottom-up, domain-independent kind of processing goes; and whether it only goes as far as shading and minimal line-finding or whether it goes as far along as symbolic labeling and aggregation of features into larger units. I do insist, however, that this boundary does not transmit or maintain a world model. I disagree with the metaphor of inverse optics, at least beyond the earliest visual processing. Surely some understanding of the physics of image formation is useful in building a vision system. But one should not take that metaphor to the extreme of saying the vision system reconstructs the world in your head by inference from the signal on your retina (*cf.* Horswill 1988).

Motor control in Pengi is very simple. In the arcade, one's left hand is on a joystick that can go left-right-up-down and one's right hand is on a button for kicking. Pengi does not model the relevant motor skills in any detail. Three wires encode the possible values passing from the central system to the game simulation.

Computationally speaking, the visual system and the central system differ in their use of massive parallelism. Both systems perform a tremendous amount of computa-

tion all the time, but they distribute their parallelism differently. The visual system's computation is parallel across the image. It performs locally connected parallel computation among a stack of flat arrays of simple elements, one array per stage of visual processing. These flat arrays generate a series of intermediate representations of the visual field. Certain kinds of global information might be collected or distributed, but relatively few, and certainly no elaborate structures are built. Once computed, these representations are all available to the central system. The central system's parallelism, by contrast, is distributed across considerations. A wide variety of matters might enter into determining the best next action. Many details might become relevant. The player might consider taking a lot of different actions. Time is short and it must continually decide. Thus the central system needs to be able to consider its options, reasons for and against them, issues that bear on its decisions, and so forth in parallel. Ultimately, of course, at the point where motor commands are issued, all that parallelism has to result in some decision—left or right, up or down, kick or not kick—so all that logic converges eventually. Nonetheless, a great deal of parallelism is both necessary and possible.

Pengi's account of the boundary between the visual system and the central system roughly follows the 'visual routines' scheme of Shimon Ullman (1984), though Pengi diverges from Ullman's account on some points. (For an important related project see Treisman and Gelade 1980.) One can, for instance, read Ullman as not requiring this sharp break between vision and the central system, only that routines run over certain representations. Though we agree with Ullman's conception of visual routines as employing a set of primitive visual operators (Ullman refers to these as *elemental operations* or *basic operations*), we differ from Ullman in how we imagine the routines themselves to be implemented. For Ullman, a routine resembles a computer program that might be either compiled on the fly or retrieved from a library. For us, by contrast, a visual routine is merely a theorist's abstraction for a common pattern of interaction between an agent's central system and visual system. The player decides on each next moment what operations to invoke based on its current understanding of its situation and options.

The boundary between the visual system and the central system is like a horizontal microinstruction set. The visual system presents to the central system a set of visual operators that the central system can invoke any time. The central system side uses familiar methods from logic design. A typical visual operator might follow a line, shade in a region, pick out the red bit and put a marker on it, or tell whether the red bit is moving. Figure C4.2 is a diagram of the general case of a visual operator's interaction with the central system. (Hardly any of the operators are this complicated.) Operators can take arguments, all of which are simple one- to three-bit quantities, *not* pointers or symbolic names or structured information. (The arguments may or may not be necessary, as we will see.) Most operators take one or two arguments, though a couple take more. Once the arguments are available, the circuitry can assert an enable line to actually perform the operation. A result is produced. There might be a number of results, but only a few operators have more than a single bit of result. Another bit, the

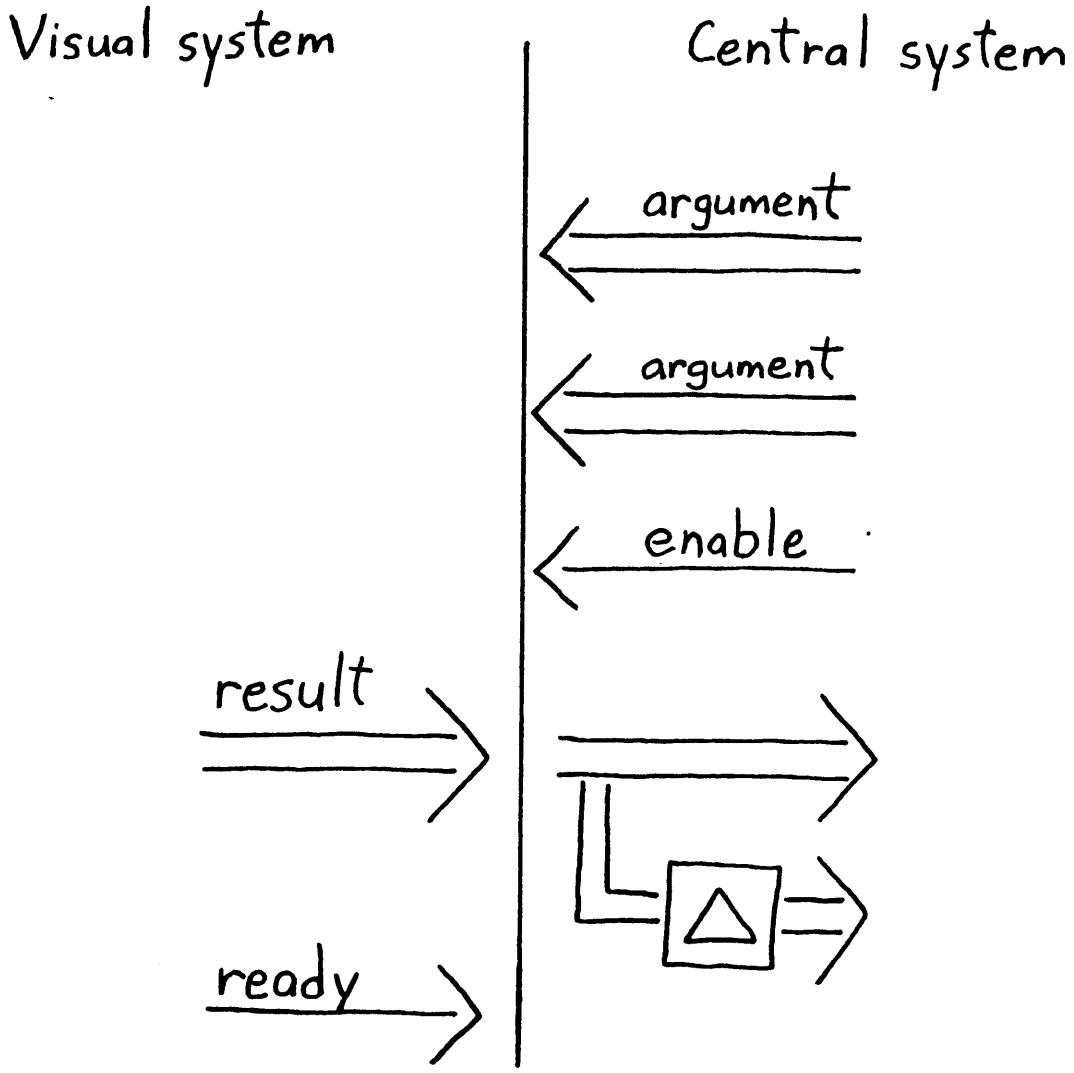

Figure C4.2. Pengi's central system interacts with its visual system through its visual operators. Many visual operators provide a constant read-out of some aspect of the visual image, but others have more complex interfaces. Some of them take arguments, all of which are carried on one- to three-bit buses. Others cause side-effects and are governed by a control line that indicates when they should take place.

mirror image of the enable bit, indicates that the result is ready. The operation might also cause side effects within the visual system.

The results that come back from a visual operator, in addition to being immediately available to the central system's circuitry, also go through delay lines. Thus the central system can use the results from both this clock cycle and the previous one. The reason for this, briefly, is that the central system has no state. It is combinational logic that is clocked on its inputs and its outputs. It has no place to remember things like what it was doing, or what large goal it is pursuing right now, or what use it is making of a given operator right now—whether the operator, for example, is referring to a bee or to spaces between ice cubes. Thus the operator values that motivated the last cycle's queries is kept around long enough for the system to determine what use to make of the answers. The solution of provided delay lines on operator results may not seem very general, but it is in accord with the principle of machinery parsimony: only postulate the absolute minimum amount of machinery necessary to cleanly explain the dynamic phenomena. One could implement the idea of 'remembering what you were doing' with latches, and for a while the network had some latches. But if a simpler scheme can, without unworkable complexity, limit the state to just a delay and corner it in one part of the system, then it is better to adopt the simpler scheme until evidence or experience dictates something more general.

Roughly speaking, the system uses a standard two-phase clock. The central system has inputs and outputs and gates in-between. (Our convention is to reckon 'inputs' and 'outputs' from the point of view of the central system, not the visual system.) When the inputs change, the clock lines on the outputs need to wait until the circuitry has all settled down before admitting the outputs into the visual system. And once these commands to the visual system are ready, they need to be driven long enough for the visual system to run. Since the wires leading from the central system to the visual system are clocked in this way, the wires going the other way—*i.e.*, the inputs to the central system from the visual system—do not need to be clocked, though in the current implementation they are anyway for clarity. Thus one two-phase clock clocks the inputs to the central system and the other two-phase clock close the outputs from the central system. The motor outputs are clocked together with the other outputs. The input clock lines also govern the delay lines.

C4c Visual system

Conceptually, Pengi's visual system has two components, early vision and the machinery implementing the visual operators. Each component is characterized by its form of computation. Early vision performs spatially uniform, bottom-up, local-based operations on the retinal image to produce a set of two-dimensional *base representations* of the visual scene. Its various modules find edges, calculate depth from stereo disparity cues where possible, infer surface normals from shading cues where possible, and so forth. The visual operators perform hierarchical aggregations of the information avail-

able in the base representations to individuate visual objects, mark patches with certain specified properties, calculate spatial relationships among the marked patches, and so forth. Pengi performs all these computations in simulation. For more about how they might be organized in a real visual system see (Mahoney 1987).

Pengi's visual system provides about twenty visual operators, depending (as we will see) on how you count. The first methodological principle governing these operators is the principle of machinery parsimony. Postulate only the minimally necessary operators and use the simplest operators that will cleanly do the job. Past that, the set of visual operators should have several other properties as well.

1. The operators should permit Pengi to play a competent game of Pengo.
2. Each operator should plausibly be computable by the sort of visual system architecture we have envisioned.
3. The operators should not strike our engineering judgement as kludges.
4. The operators should be domain-independent, in the sense that specifying or computing them should not invoke Pengo-specific knowledge.
5. Each operator's existence and behavior should be verified by psychophysical experimentation.
6. The set of operators should be sufficient and convenient for a broad range of activities, ideally including the whole of human everyday life.

Later in this section I will consider Pengi's visual system against these considerations in detail. For the moment, I should mention that whereas the first three properties are nearly satisfied, some serious deficiencies remain: the operator set is very incomplete, arbitrarily specifies several underconstrained design issues, needs far more psychophysical verification, and (as we will shortly see) contains some Pengi-specific operators in areas where the current state of research did not offer us enough guidance to formulate a serious theory. I will have many occasions to express uncertainty about the computational reasonableness and empirical validity of details of Pengi's visual system architecture; these issues all invite future work.

The fourth criterion is that the operators have to be domain-independent. This domain-independence has two aspects. The relatively easy aspect is that the operators should not depend on domain-specific facts to perform their operations. The operators should not be defined in terms of domain-specific concepts and categories. The current operator set violates this rule in a couple of places, but the violations are not important. They could easily be repaired, but the available evidence didn't sufficiently constrain them.

The second aspect of domain-independence is more subtle. The visual operator set should not be biased or restricted to some domain or some kind of domain. This condition can be hard to judge without independent verification. Ideally someone should

implement a set of visual operators that satisfies at least the first three criteria in some completely different domain. Mahoney has done some pencil-and-paper exercises with visual operators for reading topographic maps, for example (personal communication). One could imagine implementing visual operators for picking up parts or for making breakfast or for driving down a small-town street. One could then compare the results for the various activities, try to make them compatible, and combine them to make a set of visual operators that is sufficient for each individual activity and eventually for all human activity. Obviously a great deal of work is necessary before this plan can be put into action.

The fifth criterion is that the operators' existence be verified by psychophysical experimentation. If one's goal is to do engineering rather than psychology then this is obviously not a consideration. If one's goal is to do psychology then it this fifth criterion could conceivably conflict with the third and fourth criteria. That is, it could be that people have some operators that strike us as inelegant or domain-specific. Until faced with overwhelming evidence, though, one should resist such hypotheses in favor of hypotheses consistent with people being well engineered.

The visual operators are defined around three concepts:

Objects. The visual system individuates *visual objects.* A visual object is *not* a three-dimensional articulated object and Pengi's visual system does *not* perform any kind of 'object recognition.' Instead, the visual system marks off as an object, purely on visual evidence, a patch of relatively uniform qualities that is relatively localized (that is, not complicatedly distended), moves as a coherent whole, and has some kind of locatable boundaries. Several of the operators are defined in terms of these objects. The Pengo world, conveniently, has exactly three types of objects: penguins, bees, and ice cubes. The three object types are readily distinguishable, largely because the video game's designer has worked to make them readily distinguishable. It is up to the central system to invoke the operators which distinguish the various object types.

Indexing operations. Pengi has operators that permit it to pick out the "odd man out" (Ullman's phrase) in a visual scene. The indexed feature might be the only red bit in the image, or the only curved bit, or the only moving bit. The properties the visual system can pick out are called *indexable properties.* Typically only very simple, atomic properties are indexable. For more complex properties, one must perform a number of operators or even resort to serial scanning of the visual scene.

Markers. Many visual operations are predicated on a focusing mechanism, suggested by Ullman, involving *markers.* Markers implement a certain sense of focus. Using a marker, one can mark a location in the visual field for future reference. Suppose you have gone to the work of locating a place on the image where something interesting is going on. If you should then

> go off and focus somewhere else, under certain conditions you can jump quickly back to your previous location. Many operators are defined in terms of markers, especially operators for determining the spatial relationships among the locations they mark. A marker resting on a moving object moves along with it; this is called *tracking.* I don't know how many markers people have, but Pengi has six.

(We have defined our versions of the indexing and marker concepts broadly enough that they subsume the functionality that Ullman refers to as *shifting the processing focus.* Pengi has no operators for what Ullman calls *boundary tracing.*)

Before listing the operators, let us describe how the central system uses the visual operators. Each operator has a protocol of a sort that follows standard logic design practice. Most operators have both inputs and outputs.

> *Inputs.* Many operators have arguments, which are usually three-bit buses specifying markers, directions, or distances. Each operator that has side-effects has a single-bit *activation* input wire that tells the visual system to perform the operator. All the other operators run on every cycle.
>
> *Outputs.* Almost all outputs are binary signals that answer specific queries. Some operators are guaranteed to produce an accurate answer on their output wires by the next clock cycle. Other operators each supply a *ready* output wire that indicates that the output wires have been assigned a new set of values.

All the output wires are clocked so that their values remain available to the central system until they are revised with new values, usually on the next clock cycle. The input wires are clocked so that the visual system only sees their values after the central system's circuitry has had a chance to settle down after the last change of the output wires' values.

Operations on markers fall into five groups. (Keep in mind that Pengi implements all of these operators in Lisp simulations, not as real computations over visual images.)

Indexing operators.

> index-thing!(m,t): Cause marker m to move to some t (either bee, penguin, or cube), if one is visible. If it is already on one, choose a different one.
>
> index-thing-near!(m,n,t): Cause marker m to move to some t (either bee, penguin, or cube) in the vicinity of the object marked n, if one exists. If it is already on one, choose a different one.

index-moving-thing-near!(m,n,r): Cause marker m to move to a moving object within distance r of marker n. r is one of a small number of roughly exponentially increasing distances.

Marker assignment operators.

warp-marker!(m,n): Move marker m to the same location as marker n. If marker n is tracking an object, then marker m will commence tracking that object as well. (There are actually two instances of this operator, called warp-marker!-1 and warp-marker!-2. I will explain this and several other odd facts in a moment.)

warp-freespace!(m,n,d): Move marker m onto whatever object you find, if any, by starting at marker m and moving over the empty space in direction d. d is one of north, south, east, west.

unassign-marker!(m): Remove marker m, so that it no longer has any location.

Marker inspection operators.

marker-m-assigned?: Is marker m assigned? (One operator for each marker that ever needs it. Each operator is always running.)

marks-bee?(m), marks-penguin?(m), marks-cube?(m): Is marker m resting on a bee/penguin/cube?

marker-m-moving?: Is marker m moving? (If so, it follows that marker m is tracking a moving object.) (One operator for each marker that ever needs it. Each operator is always running.)

Marker comparison operators.

towards?(m,n): Is marker m moving toward marker n?

near?(m,n,r): Is marker m within a distance r of marker n? r is one of a small number of roughly exponentially increasing distances.

nearer?(n,p,q): Is the distance between marker n and marker p greater than the distance between marker n and marker q? (Pengi has two copies of this operator, named nearer?1 and nearer?2.)

direction-from-m-to-n: Returns the direction from marker m to marker n. Two two-bit outputs, representing the major and minor axes (out of the four compass directions) of the vector from m to n. This operator is replicated across several pairs of m, n. All these replicated operators are running all the time.

aligned?(m,n): Are markers n and m aligned (within some fixed, small tolerance) along either axis? If so, an output line encodes which axis. (Pengi has two copies of this operator, named aligned?1 and aligned?2.)

Object comparison operators.

> adjacent?(m,n): Are the objects under markers n and m adjacent?
>
> wholly-beside?(m,n): Is the object under marker n wholly in direction d from the object under marker m?
>
> freespace-between?(m,n): Is there nothing but empty space between the objects under markers n and m?

This list of operators reflects a series of design choices. Lacking enough empirical information, we had to make many of them arbitrarily.

It is an unresolved empirical issue how many markers people have. It is also not clear whether all markers track moving objects. (In Ullman's version of the theory they cannot.) Perhaps only one of them can, or only a few. Pengi rarely needs to track more than two.

Most of the operators take particular markers as arguments. The aligned? operator, for example, consults two three-bit input buses, each of which encodes one of the markers whose locations it should compare. I am not sure if this is reasonable. What is the engineering trade-off? The circuitry for implementing the argument scheme, while not baroque, *is* a little complicated. One might choose, instead, to provide a separate operator for every pair of markers. But this alternate scheme would seem to require many more operators than the evidence currently warrants. A compromise proposal would be to provide operators only for the markers that need them. One would like to make this assignment in a principled fashion. Perhaps some heavily-used markers have many operators whereas others are only used in special circumstances. This seems like a reasonable proposal, but evaluating it will require some experience assembling suitable operator sets for other domains.

The indexing operators take types of objects as arguments; Pengi's visual system can primitively distinguish bees from penguins from cubes. At first sight this might seem intolerably domain-specific. But in the video-arcade version of Pengo, the various types of objects are very readily distinguishable by properties that might be expected to be indexable: their colors, rough shapes, and characteristic forms of motion. Pengi's visual system can distinguish the various types primitively, but we could just as easily have arbitrarily assigned distinct colors to the different types of objects and provided a index-color! operator.

We chose to make the index! operators take the object type as an argument rather than providing separate operators for each type out of simple parsimony. The question of whether the indexing property is a parameter or whether each one has a separate operator will have to be investigated in domains where one uses a greater variety of indexable properties.

Some of the operators neither take arguments nor cause side-effects. Pengi maintains a convention that such operators have require no activation inputs and continually update their values. They might be thought of as providing a read-out of a continually

computed parameter of the image. This convention seems fairly reasonable for the particular operators in this set, but it need not be reasonable in general, depending on the practicalities of the machinery implementing the architecture. Two sets of always-running operators, marker-assigned? and marker-moving?, are replicated across all the markers.

A third set of always-running operators, direction-from, is replicated across certain pairs of markers—about half a dozen pairs. This is because Pengi often needs to keep track of the spatial relations among several moving objects at once. We have no principled reason to believe that this particular set of marker-pairs should be sufficient across all tasks. But Pengo-playing, with all its incompletely predictable moving objects, requires an awful lot of continual analysis of spatial relations compared to most activities. It is an empirical question whether Pengi can judge more spatial relations at one time than people can. Most likely it can. The machinery hypothesized for shifts in selective attention by Koch and Ullman is also more strongly focused than Pengi.

More generally, it seems more necessary to replicate operators across their possible arguments when different lines of reasoning have frequent, conflicting needs for the same operator. Often we find a trade-off between complexity of arbitration schemes in the central system and proliferation of operators. We have tried to call each trade-off according to our engineering judgement, but more experience with other domains will be required to make these choices in a more principled fashion. This is an important area for future research.

A few operators have multiple copies in cases where it seems necessary for Pengi to use the operator in multiple ways, with different arguments, during the same cycle. We don't know how reasonable this is, but we expect that the necessity is specific to time-pressured domains like Pengo. In any event, I suspect that much of the skill of playing Pengo well is in developing complex schemes for sharing operators across the various purposes to which they continually need to be put. I also suspect that some of these schemes are more complex than we can comfortably wire up by hand in building Pengi's central system.

Some of the indexing operators start "near" some already marked location. (They are related to what Koch and Ullman call *proximity preference* in shifting operations.) These operators offer no strict contract about which object they will pick out or whether it will be the very nearest, except that if they are activated often enough the operators will eventually pick out each of the objects in turn. Recent experimental evidence on *return inhibition* (Klein 1988) suggests that we would be justified in making Pengi's indexing operators cycle through all the nearby indexable objects rather than always choosing new ones randomly. This would greatly improve Pengi's performance since it would be able to focus on newly dangerous ice cubes and bees more quickly.

Likewise, the comparison operators that determine "nearness" and comparative "distance" are not guaranteed to be very accurate, and the central system should not depend on any great accuracy. Some applications would probably benefit from a more carefully specified qualitative contract, though.

Some of the operators take a "distance" argument. The argument can only take a smallish number of values which correspond roughly to the various scales on which the visual system operates. Pengi's visual system has no worked-out conception of multiple-scale vision, but I suspect that many operators are replicated across scales. Another possibility is that the whole operator set employs one scale at a time and the central system can change this scale whenever it wants. In any event, Pengi only uses one of the legal values for distance, a value corresponding to about a quarter of the width of the game board.

Some of the operators suggested by Ullman employ a notion of *bounded activation* or *coloring*. Ullman suggests that some visual operators 'color in' certain distinctive regions of the visual field. (This idea is unrelated to the perception of colored light, as in red, yellow, and blue.) This can help, for example, to determine whether something is inside or outside a curve, or to sweep out certain significant regions of the scene. It is also useful in separating figure from ground. Still, although coloring must play an important role in other domains, it has not yet found a use in Pengi. I suspect that, in Pengo-playing, coloring is only necessary for some very advanced sorts of navigation amidst mazes of ice cubes.

It is evident that the design of these visual operators is underconstrained. What is more, Pengi's visual operators do not address many of the issues in vision that do not arise in playing Pengo. Examples include texture, scale, shading, and depth. In a more realistic set of operators, the individual operators would have to be more general and there would have to be more operators for all of those concepts. Perhaps a hundred different operators are necessary, but it is far too early to tell. In the end, it is an empirical matter what operators human beings have and whether those operators would be ideal for robots. Psychophysical experiments can help resolve such questions. Ullman describes a number of such experiments.

C4d Central system

Recall that the central system is made of combinational logic. Inputs arrive from the visual system for both this clock cycle and the previous one; the circuitry generates the outputs. Some of the outputs are operator requests to the visual system; others are commands out to the motor system. The central system consists of several hundred gates, most of them specified individually in a simple language.

The hardest part of building the central system, particularly in this domain, concerns contention for visual operators. A Pengo board is a visually busy place. Often Pengi will have three different uses for some operator at once. Consider, for example, the operator that determines whether two objects are close together. Pengi might want to know if the bee is close to the obstacle and if the penguin is close to the projectile at the same time. Thus arises the difficult technical question, already mentioned in the discussion of the visual operators, of how to arbitrate among competing claims on an operator. This question has a number of answers. In building Pengi's circuitry we use a set of

macros for representing conflicts and priorities among various claims on an operator. When this doesn't suffice, as we have seen, it can become necessary to replicate the visual operators. There is a trade-off between very complex forms of arbitration and operator replication, but this is not the best domain to explore the trade-off. The real experts in the game, I suspect, are doing *very* sophisticated things to use their visual operators. They may depend on properties of their circuitry and operators that Pengi does not model, such as the slow propagation time of brain circuitry.

To read the remainder of this chapter you will have to be very comfortable with basic digital logic design ideas.

We built the circuitry with Lisp procedures; the simulator calls this code before starting the game. To build an **and** gate we call **andg**. The **andg** function's arguments are the gate's input wires; its output is the gate's output wire. The **org** function builds **or** gates; the **invert** function builds inverters. This being Lisp, we can nest these calls, making an **and** gate whose inputs are taken from the outputs of **or** gates whose inputs are taken from the outputs of inverters. The wires can be assigned to global variables or passed around as arguments. Constants include the various directions and scales that are sent to the operators; typically they will be one to three bits worth of information. Another set of functions generates simple circuitry for manipulating these buses. For example, **if-bus** takes a condition wire and two buses and returns a output bus that is driven by one of the two input buses according to the condition; this is a simple matter of logic design, two gates for every bus line. (This simple scheme for specifying circuitry is not at all novel. It resembles Kaelbling's REX language (1987).)

The code that builds the central system's circuitry largely revolves around the problem of deciding which values to place on the operators' input wires. Much depends on whether a given operator has more than one use. When an operator only has a single use one can use the **set-inputs!** form to permanently set its inputs. For example,

```
(set-inputs! marks-penguin? marker penguin-marker)
```

says to set the **marker** argument to the marks-penguin? operator to the value of the global variable **penguin-marker**, which is 0. Thus the marks-penguin? operator's result will always indicate whether marker 0 is resting on the penguin.

Canned arbitration circuitry is necessary in the more complicated cases. This circuitry is generated by a set of functions— **condition**, **overrides-action**, and several others—that expand into calls on basic circuit-construction functions like **andg** and **invert**. These functions implement much the same argumentation scheme first discussed in Chapter B4 and used in the rules in the running argument system. As in the running argument system, the metaphor is that various patches of circuitry in the network conduct an argument. A patch of circuitry can propose an action, either a primitive action (such as kicking or moving a marker) or a compound action (like engaging in a certain bee-hunting tactic). Another patch of circuitry might make its own proposal or else raise an objection to the first proposal. A proposed action is taken provided no objection against it is sustained. The **condition** function declares a condition

under which some objection should be posted against an action. The **overrides-action** declares that the taking of one action constitutes an objection to some other action. Each function defines appropriate circuitry in a straightforward fashion.

For example, the following code arranges for marker 0 to index the penguin as soon as the game begins:

```
(action index-to-penguin index-thing!
  marker penguin-marker
  object-type (constant 'penguin 'object-type)
  doit? *t*)
```

The **action** function defines an *action* named **index-to-penguin**. In Pengi's terminology, an action is an assignment of values to an operator's inputs. The **action** function creates circuitry that gates these values onto the operator's inputs on those cycles when Pengi decides to take that action. In this case the operator is index-thing!, which has three inputs: the marker to be moved (*i.e.*, **marker**), the indexable property to be employed (*i.e.*, **object-type**), and the operator's activation line (*i.e.*, **doit?**).

This next bit of code assures that Pengi only performs the **index-to-penguin** action on the very first cycle of the game. Once Pengi does perform **index-to-penguin**, marker 0 will rest on the penguin for the rest of the game. As a result, the marks-penguin? operator, whose **marker** input we set to 0 a moment ago, will return false on the first cycle and true forever afterward.

```
(condition index-to-penguin (invert *marks-penguin?-result*))
```

The **condition** function takes an action name and a wire. Here the wire is the output of an inverter whose input is the result of the marks-penguin? operator. (In general, the result of an operator is a wire or bus bound to the global variable ***operator*-result***.) The index-thing! operator is also used to find bees; another bit of code assures that the action implementing this use of the operator is suppressed while Pengi is finding the penguin:

```
(overrides-action index-to-penguin index-to-bee)
```

The **overrides-action** function takes two actions and generates circuitry that assures that the second one is suppressed whenever the first one is activated.

To summarize, the bits of code we have just discussed construct circuitry for three closely related purposes: (1) to assure that the result of the marks-penguin? operator always indicates whether marker 0 is located on the penguin, (2) to use the index-thing! operator to move marker 0 onto the penguin on the first cycle of the game, and (3) to assure that Pengi does not attempt to use the index-thing! operator to find both penguins and bees on the same cycle.

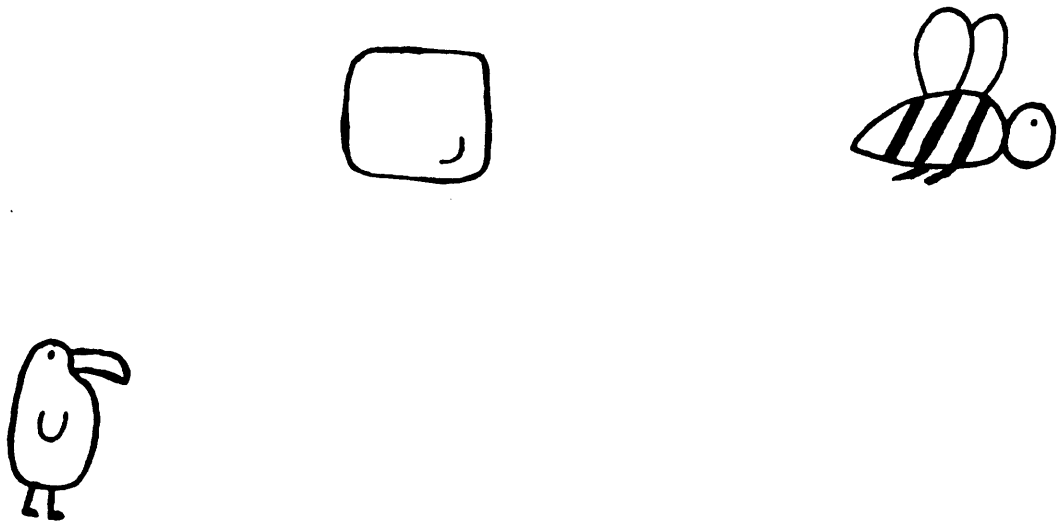

Figure C4.3. In this schematic situation from a Pengi game, the player will notice that the ice cube is aligned with the bee, move the penguin alongside the ice cube, and tell the penguin to kick it.

C4e Example

Figure C4.3 presents a schematic situation from a Pengo game. The penguin finds itself in a particular situation involving two ice cubes and a bee. (In a normal situation more blocks and bees would be found nearby.) It would probably be a good idea for the bee to walk up behind the ice cube and kick it at the penguin. How does this work? In my explanation I will interpolate some of the code, simplified in various ways, that builds the circuitry that is responsible for these routines. I won't be presenting all of the necessary code because it is voluminous and very dense, so some of the terms mentioned in the code will go undefined or described only in English.

As we saw in the previous section, Pengi has a convention that marker 0 tracks *the-penguin*. As the penguin moves across the screen, marker 0 moves along with it. Consequently, Pengi can now perform on marker 0 all the operators that are predicated on markers and thus find things out about the penguin. For example, it can perform visual operations to judge other objects' spatial relationships to the penguin and determine whether *the-penguin-is-moving*.

Another, more complicated, convention is that marker 1 is, with specific rare exceptions, on *the-current-bee*, the bee that Pengi is either running away from or attacking. In this case Pengi is going to try to attack the bee. The circuitry must maintain this invariant that marker 1 stays on the current bee. If some other bee becomes more interesting, either more dangerous or more readily attackable, then somehow marker 1 has

to get on that more interesting bee. Thus, marker 2 is constantly picking out moving objects and some circuitry is constantly checking if marker 2 marks a bee and if that bee is more interesting, by various criteria, than the bee under marker 1. (Dangerous attacking bees, for example, are more interesting than vulnerable bees that are running away.) If it is then Pengi moves marker 1 onto marker 2, unassigns marker 2, and then starts once again looking around at other bees with marker 2. As a consequence, Pengi will drop whatever it was doing to the previous bee and now set about dealing with the new bee. Marker 2's scanning routine is always proceeding in the background. Whenever the arbitration circuitry lets it use the necessary operators, the circuitry implementing this routine tries to find the most interesting bee. These operations have a relatively low priority, but they are always trying to happen.

Here is the code that drops marker 2 (*i.e.*, **moving-marker**) on moving objects near the penguin.

```
(set-inputs! index-moving-thing-near!
  tracking-marker moving-marker
  locus-marker penguin-marker
  radius (constant 200 'distance)
  doit? (andg *penguin?-result*
              (invert *index-moving-thing-near!-result*)))
```

Some care is required to make this operator run at the right times. At the very beginning of the game, it should wait until marker 0 has started to track the penguin. More importantly, it should only run every other cycle so that other operators can spend the odd cycles performing tests on the new object it has identified. As a result, it only runs on cycles when it is not returning a result.

Once Pengi finds a new moving object, here is the code that determines whether the moving object should be declared the currently most interesting bee:

```
(set-inputs! warp-marker!-1
  marker1 bee-marker
  marker2 moving-marker
  doit? (andg *index-moving-thing-near!-result*
              *marks-bee?-result*
              (definitely *nearer?1-result*)))
```

Pengi has two warp-marker! operators because we didn't have the patience to figure out how to arbitrate between their two uses. (This could be more principled. The other use involves identifying and avoiding ice cubes that have been kicked by bees.) This copy of warp-marker! moves marker 1 (*i.e.*, **bee-marker**) to the object being tracked by marker 2 (*i.e.*, **moving-marker**) provided that we have just picked out a moving object, the object is a bee, and the bee is closer to the penguin than was the previous interesting bee. One might add many more conditions.

Figure C4.4 shows the initial scene with markers 0, 1, and 2 assigned. We draw markers with polygons. Triangle is marker 0, square is marker 1, and so forth.

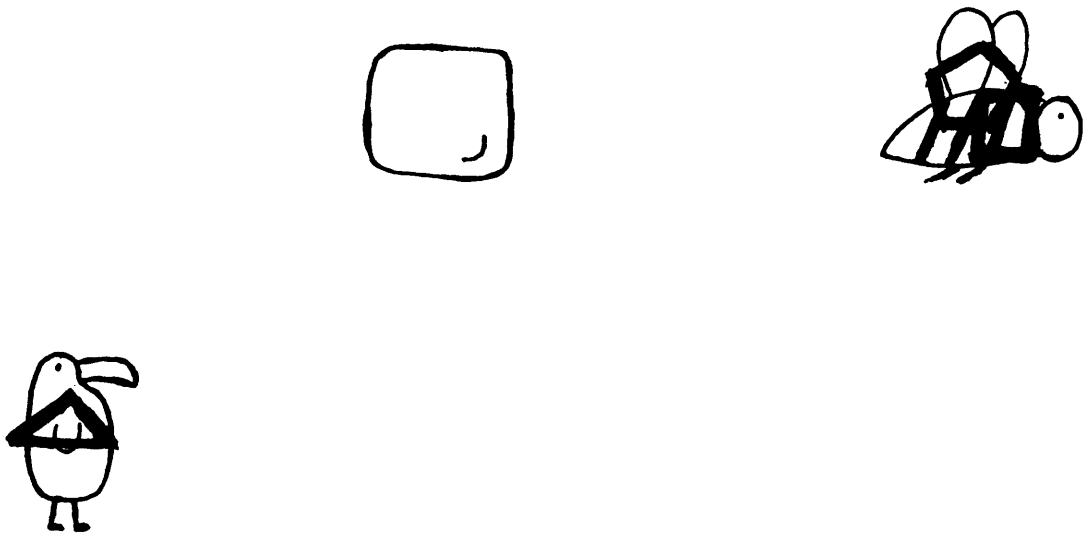

Figure C4.4. Pengi assigns marker 0 (triangle) to the penguin. It moves marker 2 (pentagon) among the nearby bees until it finds one that seems vulnerable or dangerous, whereupon it also moves marker 1 (square) onto that bee so it can focus on it.

Another operator says, pick out an ice cube near the penguin. Pengi applies this operator repeatedly, picking out all the nearby ice cubes and running some more circuitry to see if it might use the ice cube to attack the current bee. It does this by using freespace-between? to determine whether there is free space between marker 1 (the bee) and marker 3 (the ice cube). If so, and if various other conditions apply, then the ice cube is a good candidate projectile. Next Pengi starts figuring out the directions among things. It uses direction-from twice, to find the axes that relate the bee and the ice cube and also to determine the direction of the penguin from the ice cube in terms of these axes. A fair amount of circuitry sorts among the various cases. For example, perhaps the penguin will have to back up and move around behind the ice cube and kick it. Perhaps it will already be lined up with the ice cube. Different circuitry covers different cases. This case at hand is relatively simple.

Here is the code that determines whether marker 3 has managed to land on an acceptable projectile.

```
(setq *direct-projectile-won?*
  (andg *freespace-between?-result*
        (invert *direct-moving?*)
        (invert (andg (possibly *nearer?2-result*)
                      *towards?-result*
                      (invert *aligned?1-result*)))))
```

Figure C4.5. Pengi has now assigned marker 3 (hexagon) to the ice cube because it seems like a good projectile for attacking the bee.

The ice cube under marker 3 is an acceptable projectile under three conditions. First, there should be free space between it and the bee. Second, it should not be moving (*i.e.*, from someone having kicked it). (Another bit of code sets the constant `*direct-moving?*` to the output wire of a circuit that determines whether the object under marker 3 is moving.) Third, it should not be the case that the bee is closer to the ice cube than the penguin and also headed toward it. The third condition is prudent lest the bee arrive at the ice cube first and kick it.

Figure C4.5 shows the same scene with the marker 3 assigned and the various directions mapped out.

The next step is to start moving. Pengi knows what direction it needs to move now by performing a simple calculation (using logic circuitry) with the various representations of directions and axes. In the diagram, the penguin must move upward so as to align with the projectile. Once aligned, it then moves rightward toward the projectile.

Here is the code that aligns the penguin with the projectile.

```
(action align-with-projectile go!
  direction (direction-in-other-dimension-from
              *target-to-projectile-direction-major*
              *penguin-to-projectile-direction-major*
              *penguin-to-projectile-direction-minor*)
  doit? *t*)
(condition align-with-projectile (invert *aligned?i-result*))
```

The global variables used in computing go!'s `direction` input are the buses that return

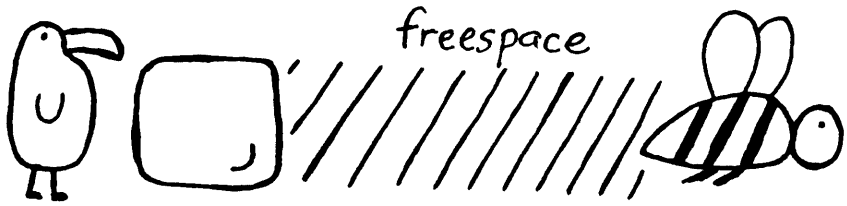

Figure C4.6. Pengi has aligned the penguin with the ice cube and is about to kick it at the bee.

the two components of the two direction-from operators. The call on the function `direction-in-...` expands into circuitry for computing the appropriate direction. Since this action is conditionalized on the penguin not being aligned with the projectile, it will stop suggesting itself once the penguin *is* aligned. Once the penguin is aligned with the projectile, the following code will propose moving the penguin *toward* the projectile.

```
(action run-for-projectile go!
  direction *penguin-to-projectile-direction-major*
  doit? *t*)
(condition run-for-projectile *aligned?1-result*))
```

Instead of the condition on the run-for-projectile action, we could equally well have specified:

```
(overrides-action align-with-projectile run-for-projectile)
```

This would guarantee that Pengi would not move the penguin toward the projectile until it had finished aligning the penguin with the projectile. The two actions are incompatible because the penguin cannot be moved diagonally.

Finally, once the penguin is adjacent to the ice cube, it kicks it at the bee.

```
(action kick-projectile kick! doit? *t*)
(condition kick-projectile *adjacent?-result*)
```

Remember that all of these operators have been running all the time. Pengi is constantly checking if there is free space between the projectile and target, and that is constantly a condition of the next move. It is also constantly checking whether the bee is aligned with the projectile, and that too is constantly a condition of the next move. Consequently, if the bee moves out of range then Pengi will stop attacking the bee, though a different bit of circuitry could make a separate decision to proceed anyway in case the bee moves back into range. This more speculative tactic is not implemented but a beginning player like me regularly does it and it would be easy to implement.

Pengi is also always checking, as much as it can, to see whether it has marker 1 on the most interesting bee. If some other bee is easier to attack or is too dangerous, it always wants to know that. That is why, as mentioned earlier, the routine whereby Pengi marches marker 2 among the various bees and compares them with the current bee is happening all the time as well. The penguin may take a definite path across the board to kick the ice cube, but the process by which it does so is not a four-step serial program. As a consequence, many different lines of reasoning want to make claims on the visual operators all the time. This is the origin of the problem of operator contention. Some of these routines can wait for a few cycles if necessary. Checking around for a dangerous bee, for example, can't wait all day, but it *can* wait for a couple cycles. It can have a fairly low priority. The programmer must make many such judgements, such as what conflicts with what and what has to happen right immediately after what.

Keep in mind that all the operators can run in parallel. It is not a matter of one operator per cycle. Still, the programmer does operate under several constraints. First, Pengi must check all its conditions frequently enough. Second, Pengi must not try to do conflicting things with an operator on any given cycle. Third, Pengi must be able to keep track of the various claims on its operators despite its minimal state. In order to interpret its operators' results, it must also be able to reconstruct what it was doing on the previous cycle. It doesn't, for example, have a queue into which it can put the different functions which it wants to assign to an operator. Nor would such a queue help, since the various operators making up a particular visual routine have to be coordinated so closely. A lot of the state comes from the markers resting on things. Pengi can reconstruct a lot of what it was doing by looking at the markers. A fourth constraint is that the programmer's task must be tractable. These considerations trade off in various ways.

The routine for kicking the ice cube at the bee can get off track for a variety of reasons. It can be a good idea to abort: Pengi might see another bee coming too close, or the bee it is attacking might move out of range, or someone might kick the projectile ice cube of the way, or something else might block its path, or something might block the penguin's path (though the penguin might be able to kick it away), etc. More generally, this routine might interact with many other routines. After all, Pengi is checking a lot of conditions all the time; many contingencies might arise. For example, Pengi might go through the first half of this routine, be distracted by something, and then come back and go through the second half. This is what it means to be continually redeciding

what to do. It can stop on a dime if something else becomes the right thing to do.

Pengi does have a small degree of inertia, so to speak. Pengi's inertia does comes not from pushing ahead with some Plan but from the inertia of its markers. The markers' being on various objects focuses the system on them and thus to a certain extent preoccupies it. Sometimes the action is moving so thick and fast that there isn't time to move marker 2 around among other candidate interesting bees, so that a bee can sneak up behind the penguin and sting it. That is how I lose the game very often. Typically when I watch Chapman play the game by looking over his shoulder, I observe by his actions that he is only noticing a portion of what I am noticing. No doubt he is seeing things that I am not as well. One's perception is necessarily very selective, and skill in the game lies largely in effective selection policies. It is not a matter of making a world model and processing everything there is to know about it. Pengi's interactions with its world have a sense of focus that arises from the interaction with its marker assignment policies with the ways in which the game board commonly evolves.

Chapter C5

Related work

C5a Context and summary

This chapter discusses briefly how Pengi and the larger project of which it is a part compare to several other projects within AI. These projects fall under two overlapping headings: cognitive architectures and theories of action as such. Since I am concerned with the relationship between theories of action and theories of machinery, I will only consider the aspects of these projects that bear on this issue. In particular, while a great deal of interesting work has been done on the processes by which Plans might be constructed (for a survey see Tate 1985), leading to ideas like *adaptive planning* (Alterman 1986), *case-based planning* (Hammond 1986), *derivational analogy* (Carbonell 1983), *meta-planning* (Stefik 1971, Wilensky 1971), and *opportunistic planning* (Hayes-Roth and Hayes-Roth 1979), my concern here is not with Plan-construction *per se* but rather with theories of action, whether they involve Plans (or even plans) or not.

Even though I have severe disagreements with most of the projects I will describe, I have learned from all of them. Because they were conducted by intelligent and thoughtful people, both their successes and their failures are informative about the nature of cognition and activity. Since many of the projects are still under way, many questions remain open.

Four themes recur. First, a theory of cognition (at least as construed mentalistically, as thought within the head), even of the cognition involved in constructing plans, is not a theory of action. Studying cognition, whether as psychology or engineering, is a perfectly valid enterprise, but theories that posit as-detailed-as-necessary cognitive models and simulations of the outside world must face the computational complexity invariably attendant on such an aggressively mentalist stance in application to real domains.

A second theme is my insistence that ideas about machinery be motivated by a serious account of activity. Toy problems cut to the measure of overly simple formulations of action offer a severely impoverished basis for evaluation.

A third theme is the trade-off between oversimplified formalizations and crippling

computational complexity that marks a great deal of AI work. Section A2d has already sketched my analysis of this problem and offered a plausibility argument for considering it an inevitable consequence of mentalist premises. Nonetheless, some recent projects have begun attacking the trade-off between oversimplification and intractability with much greater subtlety than in the past. Time will tell whether and under what conditions these projects will succeed.

Finally, several of these projects have, unlike my own, implemented their ideas on actual mobile robots. Whereas the running argument system and Pengi have merely *interacted with* simulated worlds, their robots have actually been *located in* the real physical world. Building a robot and getting it to wander down a fluorescent-lit office-building hallway, of course, is no guarantee that one's theoretical ideas are correct, but extended experience with even the most restricted robots has forced some hard thought about the hard problems of organizing activity in a world of contingency.

C5b The classical Planning literature

Several chapters have already discussed the tenets of the classical Planning literature. Activity is the execution of Plans, which are program-like structures that come from a Planner whose inputs are descriptions of the situation and goal. In Chapter A1 we saw that the very earliest definitions of Planning (Miller, Galanter, and Pribram 1960, Newell, Shaw, and Simon 1960) emphasized that a Plan is a hierarchical structure whose lower, tactical levels can often be left unelaborated until it comes time to execute them.

But much of the early Planning work was concerned with a more constrained problem: given situation and goal descriptions, produce a Plan whose execution in that situation is guaranteed to produce that goal. This early work concentrated on the issue of subgoal interactions: if a goal has a conjunctive form, as in "achieve both A and B", then the Planner must take care that the Plan steps for achieving the various subgoals don't interact destructively. The projects that pursued the problem of domain-independent Plan-construction for conjunctive goals form an impressive cooperative inquiry over a period of about ten years (Sussman 1975, Warren 1974, Tate 1975, Waldinger 1975, Sacerdoti 1977, Stefik 1980, Wilkins 1983). It's an attractive technical problem because it comes with a simple, clean formal correctness criterion. One can state "if this Planner produces a Plan then the execution that Plan in the relevant situation will effect the relevant goal" as a mathematical proposition and then prove it. As I've already mentioned, it turns out that all of this work can be viewed as proving special cases of a single theorem (Chapman 1987).

Anybody can make up a theory of action by building a machine with inputs and outputs. What's hard is to make a theory of *activity* that offers some good reason to believe that the prescribed actions will *work*. Viewing Plan-construction as automatic programming provides an especially definite reason-to-believe in the form of a correctness theorem. The problem, as Section A2f has mentioned, is that when formalizations of Plans, goals, and situations become more realistic, proving this theorem becomes im-

practical and then impossible. Some other account is required. 'Heuristic' mechanisms that merely achieve their goals sometimes are not satisfactory without a principled characterization of when they do and don't work. Nor are mechanisms that keep driving till they hit something and then figure out what to do next.

A realistic theory of activity has to characterize the larger equilibria that govern interactions between sensible creatures and benign worlds. Making a theory of activity is extremely difficult because it requires taking a broad range of issues into account. Whereas proving Plans correct tends to force oversimplification by eliminating contingency and collaboration and culture, characterizing real activity tends to force one to take all these meliorative influences into account. Technical questions tend to become continuous with philosophical and psychological and sociological questions. Much as one might wish it were different, that's how it is.

One of the most important early Planners was the Strips Planner used by the Shakey robot at SRI (Fikes and Nilsson 1971, Fikes, Hart, and Nilsson 1972, Nilsson 1984). Having been motivated by a real mobile robot rather than simulated blocks worlds, this project exhibited some uncanny insight into the nature of Planning, little of which was followed up by later work (though see Chien and Weissman 1975).

> One of the novel elements introduced into artificial intelligence research by work on robots is the study of execution strategies and how they interact with planning activities. Since robot plans must ultimately be executed in the real world by a mechanical device, as opposed to being carried out in a mathematical space or by a simulator, consideration must be given by the executor to the possibility that operations in the plan may not accomplish what they were intended to, that data obtained from sensory devices may be inaccurate, and that mechanical tolerances may introduce errors as the plan is executed.
>
> Many of these problems of plan execution would disappear if our system generated a whole new plan after each execution step. Obviously, such a strategy would be too costly, so we instead seek a plan execution scheme with the following properties:
>
> (1) When new information obtained during plan execution implies that some remaining portion of the plan need not be executed, the executor should recognize such information and omit the unneeded plan steps.
>
> (2) When execution of some portion of the plan fails to achieve the intended results, the executor should recognize the failure and either direct reexecution of some portion of the plan or, as a default, call for a replanning activity (Fikes, Hart, and Nilsson 1972, page 268).

The system that executed Shakey's Plans, called Planex, had access to a summary of the reasoning behind the Plans in the form of the Planner's 'triangle tables'. This information allowed Planex to perform some limited kinds of execution-time improvisation and repair. For example, having discovered that some plan step's precondition was already

satisfied in some way, it would know enough to cancel the plan steps whose purpose was to satisfy the precondition in some other way. But the Shakey project did not take the logical step of effacing the distinction between Planning and execution altogether, and later work in the field concentrated on Planning to the near-total exclusion of execution. A few executives grew relatively sophisticated, especially that of another mobile robot called Hilare (Giralt *et al* 1984), but clear lines have separated the executive's job from the Planner's.

Incidentally, the word 'planning' has a second, less common use in the AI literature that should not be confused with the more established use this section has been discussing. This alternate use begins with GPS (Newell, Shaw, and Simon 1960). GPS has two heuristics for searching its problem spaces (or, to use the 1960 vocabulary, 'task environments'—note that the 'task environment' is a structure in the agent's mind, not its actual physical environment). One (pp. 259-261) is the technique of means-ends analysis using difference reduction (*cf.* Section C3e). The second, called 'planning' (pp. 261-262), involves constructing a coarser problem space by, for example, systematically removing information from each of the states and operators in the first space. A solution in the coarser space might provide signposts for a solution in the original space. Although Miller, Galanter, and Pribram used many ideas from GPS in their classical formulation of Planning (1960), their actual notions of Plans and Planning are not related to those of GPS. The GPS notion of planning survives in projects such as (Durfee and Lesser 1986). Abstrips (Sacerdoti 1974) uses a closely related technique to control search in Plan spaces.

C5c Extended Planning schemes

Several projects, most of them still in progress, have sought to extend the concept of Planning in various ways. They are a disparate lot, but the ideas about activity that Section A1a called the 'planning view' continue to unite them. On this view, activity is fundamentally a matter of constructing and executing Plans. All of these projects, avowedly or not, also continue to treat Plans as computer programs in accord with the research project originally defined by Miller, Galanter, and Pribram.

Perhaps the culmination of the classical Planning tradition is Wilkins' book *Practical Planning* (1988), which describes a nonlinear Planner called SIPE. SIPE's strengths and weaknesses are succinctly described in Chapter 2 of his book. Briefly, Wilkins faces head-on the necessity for a "balance between epistemological and heuristic adequacy" that he accurately sees as the central problem for classical Plan-construction research, "retaining as much expressive power as is practical, yet making enough restricting assumptions so that a viable, efficient implementation can still be realized" (p. 12). Whereas I consider this balance unachievable for purposes of living everyday life, it is impossible to make a knock-down argument that no acceptable balance can be found. Wilkins' strategy is to reduce the complexity of evaluating the Planner's truth criterion by embracing the very restrictive Strips state-space style of action represen-

tation. He then attempts to compensate for the resulting expressive difficulties with a collection of reasonably principled heuristic methods, both domain-independent and domain-specific. Wilkins also asserts that search should be controlled with domain-specific heuristics, though it remains to be seen what it takes to tame the vast searches generated by an "epistemologically adequate" Planner in real domains. Wilkins neglects many epistemological questions by permitting SIPE to rely heavily on an accurate and continually updated world model. (It is unfortunate and telling that he uses the word 'epistemological' to refer to the expressive power of the Planner's representation scheme rather than to the practicalities of stocking its world model with definite knowledge about the actual world.) SIPE *can* express its uncertainty on specific points, but it has no principled way of compensating for its uncertainty. Thus an application of SIPE to real domains will require facing the difficulties of making genuine world models. Wilkins assumes the classical partition of responsibilities between Planner and executive. His executive is capable of initiating re-Planning when things go wrong (Wilkins 1985), but in doing so it too relies heavily on the world model. In Chapter 12 Wilkins briefly mentions a project to reconcile 'planning' and 'reactivity' by loosely coupling SIPE with PRS (Georgeff, Lansky, and Schoppers 1986) to drive the Flakey robot. This project should be very interesting to watch as it begins to seriously address the executive's responsibility to act flexibly on the spirit of Plans rather than inflexibly on their letter. To date, though, Wilkins' accomplishment has been to work within the classical Planning framework, hoping to alleviate its serious shortcomings by amending and generalizing it. While he has made classical Planning into a more serious proposition, he has also made it much harder to evaluate. In the end, we cannot seriously address whether and when and why a scheme like Wilkins' will suffice until we move on from isolated toy problems to a much more detailed theory of activity as a whole.

The recent work of Dean represents another assault on the trade-off between epistemological and heuristic adequacy in Planning. Dean's principal contribution has been an unprecedented subtlety in the application of the tools of computational Complexity theory to research in Plan-construction. Noting the seeming impossibility of resolving the trade-off between epistemological and heuristic adequacy in domain-independent Plan-construction, Dean is investigating Plan-construction in classes domains whose formalizations have some non-trivial structure. For example, (Dean and Boddy 1987) investigates the possibilities of algorithms that perform inference tasks over a formalization of causality, buying tractability at the price of incompleteness. Future work must determine whether these incomplete algorithms suffice for Plan-construction problems in real domains. (For an analogous project exploring the strategy of purchasing tractability with incompleteness in deduction tasks involving objective representation schemes see Levesque and Brachman 1987.) In a related project, (Dean 1987) investigates time-pressured Plan-construction in a formalization of action related to those of the process control literature. Instead of meaningless the 'state spaces' of classical domain-independent Planners, Dean approaches the problems of resource allocation in scheduling activities involving continuous flows of materials. Insight into the relation-

ships between epistemological and heuristic adequacy in such restricted but interestingly structured domains will doubtless lead to a deeper understanding of the issues involved in using Plans to organize other, less restricted forms of activity.

Section B4b has already discussed the concept of interleaved Planning and Section B4c has discussed the concept of postponing explicitly manipulating the recursively decomposed goal-structure.

Much work has gone into generalizing Planning's formalizations of action beyond the classical Strips assumptions of discrete operators and states, particularly by explicitly representing and reasoning about the temporal relationships among the salient actions and world states (see particularly Allen 1981, Allen and Koomen 1983, Dean 1985, Dean and McDermott 1987, McDermott 1982, Shoham 1987 and 1988, Vere 1983). As the analysis of Section A2d would suggest, the increased realism of world modeling has led to grave Complexity difficulties. Lozano-Pérez, Mason, and Taylor (1984) have begun a project to extend classical Plan-construction techniques to employ relatively sophisticated formalizations of spatio-temporal relationships motivated by problems of motion Planning in robotics. See Donald (1987a and 1987b) and Erdmann (1987) for further development of these ideas. Once again, increased realism in Planning is bought at the price of grossly intractable Complexity. See Canny (1987) for further insightful analysis of the Complexity of motion Planning and related problems. Lansky (1987) is also exploring Plan-construction Problems whose solution can benefit from knowledge of domain geometry.

Georgeff and Lansky (1987) and Firby (1987) describe two closely related projects under the rubric of 'reactive planning'. Although this phrase has been widely used in a generic way in the AI community since around 1984, for these projects it refers to a technique whereby specified conditions cause an agent to retrieve certain Plans from its library and begin executing them. The 'reactiveness' is thus fairly coarse. This technique will have to be evaluated in terms of properties of the Plan library and the forms of interaction to which the agents Plan-selection policies give rise in various kinds of worlds. (For an earlier project that refers to its techniques as 'reactive' and prefigures some of these ideas see Fox and Smith 1984a and 1984b.)

Schoppers (1987) describes 'universal plans'. A universal plan is a highly conditionalized Plan built by exploring all possible paths through the space of possible interactions between an agent and its environment. This approach of explicitly representing the dynamics of the agent's prospective activity is praiseworthy, but a representation of dynamics at the classical state-space level would not seem to have enough structure to keep the space of possible interactions from exploding and resulting in unwieldy Plans. Future work might explore the possibility of using a more sophisticated formalization of action to control the explosion by making explicit the domain structure that a state-space representation collapses.

C5d Production systems, ACT*, and Soar

In the 1960's, Newell and Simon initiated a research project that envisions using production systems to explain what people do when they pursue such intellectual tasks as constructing logic proofs and solving cryptarithmetic puzzles. Their classic description of production systems is (Newell and Simon 1972).

Simplifying a little, a classical production system is organized around a distinction between short-term and long-term memory. Short-term memory is a limited-capacity store of symbols. The knowledge in one's long-term memory takes the form of a collection of productions. When the left-hand side of a production matches the contents of the short-term memory, it fires and deposits the contents of its right-hand side into the short-term memory. Only one production fires at a time. If several productions wish to fire then some conflict resolution scheme chooses among them. Many standard control structures can be implemented within this scheme (Newell 1973) and Nilsson's (1980) text formalizes many AI issues within a production system framework.

Production systems have many attractions. They are simple and uniform. They offer a simple account of learning as incremental accumulation of productions. They suggest both the elements of flexible decentralization and parallelism of human thought (in the process by which productions decide whether they would like to fire) and its elements of rational centralization and seriality (in the selection among the candidate productions). And they lend themselves to modeling the verbal reports of subjects in experiments such as those reported in (Newell and Simon 1972).

Chapter B3 has already distinguished between the imperative semantics of productions and the so-long-as semantics of Life rules and dependency networks. Whereas both a Life rule and a production will fire when they are applicable, only a Life rule's consequences will be retracted automatically when they are no longer justified. As a result, the running arguments system always has a clear, consistent sense of what it is doing. Similar comments apply to Pengi's combinational logic, each of whose gates only keeps producing a given output so long as its inputs retain the appropriate values. Each moment's argument structure provides the system with freshly thought-out grounds for its actions.

Pengi, likewise, does not have a short-term memory, at least in the same sense as a production system. It does have its visual markers, but visual markers are far more restricted than a production system's short-term memory in that they are tied to a specific modality and cannot contain arbitrary symbols. Production systems have generally referred to individuals in the world through their corresponding names, but I don't know if this practice is part of the essence of production systems. (Some authors' examples of production systems, such as Winston's textbook example of toy productions for identifying animals (1984 pages 177-180), have employed a vaguely deictic vocabulary, but production systems have never, to my knowledge, been associated with an explicit, thought-out indexical theory of representation.)

While a production system is capable of perception (into its short-term memory)

and of action (through special clauses in productions), production systems are correctly advertised as models of cognition, that is, of abstract thought. In this way, production systems are as strongly mentalist in approach as Pengi is interactionist. Cryptarithmetic is not very much like making breakfast. This in itself certainly does not invalidate the research program. But the various production system models of cognition would seem to predict that people can solve difficult cryptarithmetic puzzles (and derive complex syllogisms *etc.*) in their heads. Plainly this prediction is false. Those people who can solve cryptarithmetic puzzles do so using scratch paper. Some people do some of their thinking while staring into space, but scratch paper is used in practically all forms of intellectual endeavor. Interaction with scratch paper may not require deftness or brawn, but it is still a complex form of interaction with the physical world whose dynamics ought to be investigated. People must use scratch paper for a good reason; what is it? The assimilation of scratch paper to short-term memory reflects a failure to acknowledge the difference between internal representations and external reality. Newell and Simon's mentalism thus extends beyond their choice of metaphors and strategies and into the untenable view that the world might as well be located inside one's head, much as the general discussion of Chapter A2 has anticipated. This complaint will become more consequential as we discuss the further development of production system technology.

Production system technology has developed continually and has been widely applied (Davis and King 1975, Bachant and McDermott 1984). The OPS5 system (Forgy 1981), for example, supported research by several groups, both as a framework for cognitive models and as a general programming language (Brownston *et al* 1985). Since the inner loop of a production system is its pattern matcher, OPS5 featured a heavily optimized pattern-matching scheme called Rete (Forgy 1982).

Various advanced architectures have been based on production system ideas. One of these, called ACT*, was developed by John Anderson (1983a). Anderson started by positing that the human cognitive architecture consisted of a production system and a semantic network. Through an impressively large program of experiments, he then attempted to determine the details of rule semantics, propagation patterns in the network, and the like. ACT* is a model of "higher cognitive processes" (p. 1), not of action. It is also very complicated and I will not attempt to describe it in detail. Anderson offers some example production rules for transforming goal hierarchies in response to the sorts of difficulties cataloged by Sacerdoti (1977), but does not discuss what one does with a plan.

Another, rapidly expanding production system project is the Soar architecture, developed by Allen Newell together with two of his students, Paul Rosenbloom and John Laird. Soar is a substantial revision and extension of the classical production system scheme. The most important revision is that the elimination of conflict resolution. When Soar runs productions, all productions that match the short-term memory run in parallel until the system quiets down. The extensions involve three ideas: the idea that all cognition involves search in *problem spaces*, the concept of *universal subgoaling* (described in Section B4d), and the *chunking* mechanism (described in Section B5f).

The Soar architecture starts from a theory of problem solving based on search in problem spaces, an idea descended from GPS (Newell, Shaw, and Simon 1960, Newell and Simon 1963, Ernst and Newell 1969). A problem space is a directed graph whose nodes are 'states' and whose arcs are 'operators'. A search is successful when some chain of operators reaches a goal state. A path through a problem space is, in effect, a simulation of some course of events in the relevant domain. The domain might be out in the world or it might concern Soar's own internal processing. If the domain is out in the world, then the search only 'solves' the actual concrete problem in the sense that it has located a Plan it expects would solve it if executed in the appropriate situation. Knowledge encoded in productions guides both choice among problem spaces and search within particular problem spaces. When these productions run, they assert preferences about how the search should proceed. If these preferences add up to a single consistent answer, that answer is accepted. Otherwise the system declares an impasse and forms a subgoal to resolve it.

The great virtue of this scheme lies in the interactions between universal subgoaling and chunking. Universal subgoaling is a uniform control scheme for every decision the system ever makes. Likewise, chunking is a uniform learning scheme for summarizing the outcome of any decision process the system ever successfully undertakes. Other research projects have used problem spaces for both problem solving and learning (*cf.* Mitchell 1983). The novelty of Soar is the elegant, general way in which it uses problem spaces. Through its *aspect generality*, Soar avoids, to use Mitchell's wonderful phrase, the "wandering bottleneck problem."

Chunking is billed as a "general learning mechanism." Evaluating this characterization is a tricky matter. It is general in the sense of being domain independent, but even this claim requires problem spaces to be an appropriate way to organize activity in all domains. Chunking does accelerate Soar's problem-solving, but as with the analysis of dependency maintenance in Chapter B5, one cannot conclude that a chunk merely speeds up the system's performance in the exact same situation. Instead, recurring patterns of transfer cause the system to exhibit interesting forms of apparent insight into the problem. A chunk formed for a particular subgoal will transfer to anywhere that subgoal appears, whether in the same task or in different tasks.

Furthermore, because chunks use variables to abstract away from situation particulars, they can transfer to analogous situations as well. Laird, Rosenbloom, and Newell describe a particularly striking example in which the within-task transfer of a chunk implicitly exploited problem symmetries to avoid a threatened exponential search explosion (1984). Chapter B5 speculated that deictic representation would permit dependencies a similar generalizing power. It is hard to make a direct comparison, though, given the radically different contexts in which Soar's chunking and the running argument system's dependency maintenance operate. Certainly, as Chapter C3 has argued, deictic representation ought to generalize across instances of deictically specified subgoals, abstracting across instances passively instead of using variables and pattern matching. For example, a particular patch of circuitry permits Pengi can pursue the subgoal of

kicking an ice cube in the context of several different larger goals. And if Pengi did any higher-level arguing such as that demonstrated in Section B4d then the higher-level arguments would transfer passively as well. But Pengi does not have a general problem space scheme. It does not need one, but then Pengi is not trying to solve puzzles in its head.

Soar has only recently been fitted with machinery for perception and action (Rosenbloom, personal communication). In the published examples in which chunking has been demonstrated, the action takes place entirely within Soar. As a result, everything chunking learns can be derived from what the system knows when it is set running. One might argue that, because it does not introduce any previously underiveable information, chunking is only a speed-up mechanism, and while a general speed-up mechanism is certainly a fine thing, it could not cause Soar to solve any new problems. While I believe this argument to be correct in spirit, it is not quite accurate because chunking can overgeneralize and is thus unsound. One might regard the unsoundness of chunking as a positive heuristic rather than as a simple violation of logic. An overgeneralized chunk might lead Soar to explore regions of its problem space it might not have explored otherwise, thus possibly achieving new or better solutions to subsequent problems (*cf.* Anderson 1983b, Laird 1988). I do not know if the Soar group has investigated the utility of this heuristic in realistic domains. As with any overgeneralizing induction scheme, the domain representation will determine which overgeneralizations chunking makes. Consequently, the utility of overgeneralization as a learning heuristic depends strongly on the suitability of the representation. This is not an objection in itself, but it is another reason to use good representations.

In any event, Soar does not learn anything about the outside world and chunking offers no obvious guidance about how such learning might proceed. As my analysis of the running argument system in Chapter B5 demonstrated, simple access to the outside world does not suffice if 'access' means that the agent magically maintains an accurate world model. Fortunately, though, Pengi and Soar both have somewhat more realistic perceptual systems. I expect that chunking will prove to have a role in a model of learning about the world, just as I expect that dependency maintenance has a role in routine evolution. But both propositions have yet to be properly worked out and demonstrated.

When Soar is asked to participate in complex activities in realistic domains that involve an outside world, it will have to face all the hard questions of what Wilkins called epistemological and heuristic adequacy. The states and operators of problem spaces are highly analogous to the Strips model of action and make a very poor model of actions in the real world. Likewise, the states in a state space look tractable enough they represent positions in an 8-puzzle, but when they represent configurations of a kitchen they face all the difficulties of building realistic world models. In approaching issues of activity in the real world, the Soar project must search for a set of restrictions and extensions that will strike an acceptable balance between expressive power and efficiency. The work of Wilkins, Dean, and others represents a practically identical

search that has been long under way in the Planning literature. As with the Planning literature, some strong heuristic arguments suggest that no such balance can be found. But, also as with the Planning literature, no knock-down arguments are possible. Much will be learned from each search.

Beyond my detailed objections to Soar's architecture is my disagreement with problem solving as a view of activity. The problem-space formalization of states and operators would seem appropriate for domains such as theorem-proving and puzzle-solving that are actually characterized by discretely individuated world states produced by a series of discrete actions taken from a definite set of types. These stipulations are extremely restrictive. Many activities, most notably real physical activities in kitchens and offices and garages, exhibit more complex forms of interaction.

C5e Situated automata theory

In work over the last several years, Rosenschein and Kaelbling have been developing *situated automata theory* and applying it to the construction of mobile robots (Kaelbling 1988, Rosenschein 1985, Rosenschein and Kaelbling 1986). Situated automata theory envisions a robot controlled by a digital logic circuit. The theory provides two tools for analyzing the robot's relationship to its surroundings.

The first tool is a model-theoretic semantics for a modal logic of knowledge that permits a formal analysis of the content of—in their words the "information carried by"—any given region of the circuit, down to a single wire.

The second tool is an algorithm for deriving such a circuit from a theory expressed in their logic. This algorithm first builds an abstract circuit by backward-chaining from all possible actions through the theory's space of possible inferences. Then it uses standard logic-design methods such as constant-folding to make this circuit manageably compact while preserving enough information about its provenance to permit ready inspection and debugging.

These tools have been implemented and demonstrated on a simple mobile robot called Flakey. More amibitious demonstrations are currently under construction. The situated automata project invites comparison to my own insofar as both offer technical and theoretical prescriptions for the design of situated agents. A careful comparison, though, shows that the two projects are based on conflicting philosophies and have significantly different though partially compatible aims.

First let us consider the situated automata theory of the information content of machinery states. Situated automata theory provides the user with a notation. This notation is a modal generalization of first-order logic and has a sophisticated model-theoretic semantic theory. This semantic theory's basis in model theory marks the situated automata notion as an objective theory of representation. The particular semantic innovation of the theory is to stipulate that an agent's sensor readings provide points of space-time at which models of the theory an agent embodies can be pinned, as it were, to the agent's concrete circumstances. The sensor readings enter the the-

ory as axioms stating that certain readings were obtained at certain points in time. In other words, the semantic theory relates the causal contacts between an agent and its surroundings via its sensors to a correspondence theory of the truth status of its representations.

Whereas situated automata theory provides a theory of *knowledge*, I have been presenting a theory of *activity*. When designing the machinery for a situated agent of any sort, there is no substitute for an understanding of the dynamics of the agent's intended activity. Situated automata theory provides a theory of action which formalizes the proposition that the robot will take the actions that it deduces it should take. Although a theory of action for one's agent is indispensible, it is still only a small part of a proper theory of activity. An understanding, even a limited one, of the complex relationships between machinery and dynamics, can motivate a principled methodology for the design of situated agents. Once one has designed particular machinery, the formal methods of situated automata theory might provide insights in the detailed analysis of certain aspects of the machinery's interactions with its world, namely what information various elements of machinery carry under various conditions. The value of any such insights, though, would seem to be contingent on the value of the concept of knowledge in thinking about situated agency. Let us investigate the matter in more detail.

While Rosenschein and Kaelbling certainly do not consider themselves to be doing philosophy, their theory is an instance, albeit the most credible one, of the old and perpetually unsatisfying philosophical view of the nature of knowledge whose first recognizably modern statement is found in Locke. This tradition seeks to explain the human ability to get along in the world in terms of propositional knowledge possessed by individuals. Its project is to explain how human beings can derive knowledge of the world from sense impressions. Insofar is it focuses on quasi-mechanical processes occurring within individuals and on structural correspondences between knowledge-within and the-world-without, this is a mentalist account of human existence—and, by extension, of the existence of any other creature or automaton to which it might be applied. Despite a proliferation of increasingly sophisticated variations, this account has come down nearly to the present day in Anglo-American philosophy without essential change.

The difficulty with basing a theory of action on a theory of knowledge is that knowledge of the world is very hard to come by. Situated automata theory, like any other theory of knowledge, must face the fact that sensor readings are noisy and do not provide sufficient justification for knowledge. Sense impressions underdetermine the world because sensors can only register that information which is contained in the energy of the appropriate kind that happens to reach them. That Rosenschein and Kaelbling's logic is monotonic, as are all logics whose model theories are satisfactorily understood and not intolerably pathological, prevents them from expressing, much less resolving, the inherently heuristic nature of sensor interpretation. That knowledge appears both necessary and unobtainable is the fundamental and abiding epistemological paradox of mentalist, correspondence theories of knowledge. A detailed understanding of its consequences for the situated automaton project will become possible as the project

proceeds. Rosenschein is currently addressing these difficulties by generalizing his logic to incorporate probabilities (personal communication). Time will tell whether he can find a principled and practical way to manage the mass of *a priori* probabilities upon which the accuracy of non-trivial probabilistic reasoning depends.

One principled attitude to the underdetermination of knowledge by sense impressions is contained in the methodology of David Marr, which advises postulating 'constraints' on sensor interpretation processes. Constraints should ideally derive from natural properties of the domain. Typical constraints in visual perception include assumptions of rigidity and general position. Constraints do not solve the problem of underdetermination from a logical point of view, though from an evolutionary point of view they might confer sufficient survival value. Attempts to generalize Marr's methodology to more general reasoning tasks have foundered on the difficulty of 'encapsulating' central-system functions sufficiently to permit the formulation of reliable constraints. For a clear exposition of the problem see (Fodor 1983).

The epistemology of a situated automaton's states is to be distinguished from that of deictic representation, which does not involve any notion of correspondence but rather a description of the causal relationships between the agent and the materials and equipment in its environment. Our explanation of Pengi's competence at Pengo does not rest on a calculation of the truth conditions of Pengi's putative knowledge of its world. Rather, Pengi's design rests on our understanding of activity in general and the particular activity of playing Pengo. Furthermore, while the semantic ideas of situated automata theory provide a certain account of indexicality in representation, it offers no particular account of functionality. Some notion of functionality is probably compatible with situated automata theory, though, insofar as functionality is best considered a dynamic concept.

Let us summarize the argument. The situated automata theory of representation is certainly more formal by far than any existing deictic theory. And this formality aids certain analyses. Nonetheless, situated automata theory does not help us understand Pengi or its representations because its monotonicity and its model-theoretic semantics are wholly incompatible with the theory behind Pengi. Pengi's theory of deictic representation was strongly guided by the special requirements of a theory of activity and is, for the reasons Chapter C3 has presented, far better suited to the purpose than an objective representation scheme such as that of situated automata theory. In designing a situated agent such as Pengi, there is no substitute for a theory of the dynamics of the agent's activities. Only on such a basis can one properly justify a theory of the machinery of situated agents. A model-theoretic semantics is not a theory of dynamics.

Finally, let us consider a few topics concerning the architecture envisioned by situated automata theory and embodied by the Flakey robot. Many people, observing that Pengi networks and situated automata both use combinational logic, have asked if one could we view Pengi as a particular situated automaton. The answer is yes. That the use of digital logic should be held to suggest any deeper affinities, though, is a remarkable measure of the theoretical disruption that the concept of software has caused in

AI research. Digital logic is simply what computers are made of. Any robot built with our current computational technology would employ digital logic, at least in its central system. Any computational theory of activity must explain particular configurations of an agent's digital logic in terms of the dynamics of the agent's activities.

One minor difference between Pengi's digital logic and a situated automaton's is that whereas Pengi's central system has no latches, a situated automaton will typically have quite a lot of them. This is not a very important distinction, though. As I've mentioned, I certainly believe that people have state in their heads. Still, the principle of machinery parsimony suggests that state be minimized, other things being equal.

Aside from machinery parsimony, a second reason why Pengi's central system does not keep any state is that 'state' in AI research has almost always meant 'world model'. Given the objective semantics of its states, a situated automaton's collection of latched bits precisely implements a world model, however dispersed. Each bit indicates a point of correspondence between the logical theory embodied by the automaton and objectively defined states of affairs holding in the outside world. Previous chapters have already articulated my dissatisfaction with world models and correspondence semantics. It is difficult enough to explain the difference between state and world models that we were glad for an opportunity to avoid keeping any state.

C5f The MIT mobile robot group

By far the most similar project to my own is that of Rod Brooks and his group at MIT (Brooks 1986a, 1986b, and 1987, Brooks and Connell 1986). Brooks *et al* have built a series of mobile robots that engage in sensibly organized interactions with their world despite their simple machinery (Brooks, Connell, and Flynn 1986, Brooks, Connell, and Ning 1988). Brooks refers to these robots as *Creatures*. His intention is not to reproduce human activity but rather to get at the essentials of interactions between very simple devices and the physical world.

Brooks rejects the conventional modularity of perception and Planning and execution. Indeed, he opposes any modularity. Instead, he proposes a method, based on the idea of *subsumption architecture*, for relating a dynamic analysis to a design for the agent's machinery. Specifically, he suggests sorting a Creature's way of life into a set of *behaviors* and building a *layer* of machinery corresponding to each. The machinery in each layer is a circuit wired up from simple, standardized components. As the word 'layer' suggests, some of the behaviors are to be considered more fundamental than others. For example, a Creature's bottom layer might lead it to wander around stochastically and avoid running into things. A higher layer might lead it to investigate any interesting objects it comes across. A yet higher layer might lead it to systematically explore a territory. Each layer operates by overriding specific control signals in the layers below under particular circumstances. For example, the object-investigating layer will lie dormant until a sufficiently interesting object comes into view, whereupon it will override the mechanism that stochastically reorients the robot, substituting its

own orientation control signals.

In building subsumption-architecture robots, Brooks *et al* use many of the dynamic principles that I have found important in my own work. Far from executing a Plan, the robot employs all its circuitry in continually redeciding what to do. Decisions leading to action are based, as far as practicable, on sensor readings rather than on world models. When building a robot that is supposed to carry out a set procedure, they permit downstream actions to be cued by the noticeable effects of upstream actions rather than by a program counter, thus allowing for great flexibility and adaptability in the actual organization of the activity on particular occasions (Connell 1987, 1988, and forthcoming). And they make a policy of avoiding unnecessary generality and keeping their machinery as simple as possible.

The ideas of Brooks *et al* about perception are similar in spirit to my own, though they are quite different in letter. The idea of visual routines rejects the idea that the job of perception is to build world model and replaces it with the idea that the agent's ongoing activities determine which aspects of the current environment are worth registering. But this idea does not extend to early vision, which is still domain-independent. For Brooks, though, the design of a given Creature's entire perceptual systems should start from an understanding of the way of life in which the Creature is to participate. Even the most basic operators applied to the retinal inputs—indeed, even whether the Creature has retinas—depends on what information the Creature needs to register under what conditions. This disjunction in our two approaches to perception is readily comprehensible in terms of our differing goals. Whereas I wish to understand human beings in their relative generality, Brooks wishes to build highly specialized insect-like robots. The principle of relating machinery to dynamics is the same in each case.

The work of Horswill (1988) develops this theme of dynamically informed visual-system design. Horswill demonstrates a mobile robot whose vision machinery can be very simple because he has understood the dynamics of the robot's assigned task, namely chasing things in an indoor environment. Rather than building a general-purpose visual system, he watched kittens chasing things, analyzed the dynamics of the activity (wandering, noticing something, moving to keep it in view) and observed that the necessary sensory information must only be registered under particular conditions. As a result, a set of very simple visual computations complement one another to assure that the robot continues participating in the dynamics of chasing. Though too simple to draw very firm conclusions, this exercise does suggest a method for designing visual systems in the context of a dynamic analysis of a Creature's whole way of life.

In another project in this group, Viola (1988) used computer simulations to explore the possibility of sophisticated subsumption-architecture Creatures arising through evolution. Surveying the subsumption-architecture design practices that had developed in Brooks' group, Viola observed that most of the robots' circuits could be viewed as assemblies of standardized circuit design clichés. Having devised a notation for a mobile robot's "genetic code" that could specify such assemblies in a natural way, he began simulating the interactions of robots with simple environments, determining in each

case the robot's success in acquiring nourishment. Robots that kept themselves alive were permitted to reproduce through a process of genetic replication that often suffered mutations. Extended simulations through several generations led to robots that participated in reasonably interesting forms of interaction with their environments. The lesson of Viola's project is strikingly parallel to that of Lenat's AM system (1982), where the 'density' of interesting representations guaranteed that a sufficient number of mutations would lead to interesting results (Lenat and Brown 1984). The interest of this work is the way in which evolution connects machinery and dynamics. By focusing on the bottom line of reproductive success, evolution can select for the adaptive value of a Creature's entire way of life without having to be smart enough to formulate a representation of the dynamics of the Creature's activity. Whereas several previous projects have used metaphors of evolution through natural selection in models of abstract cognition (Holland 1975, Lenat 1983), Viola applies this theme to actual interactions between Creatures and their world. My own notion of routine evolution (see Section B2e) uses a metaphor of evolution in describing some of the dynamics of individual learning. It would be interesting to apply metaphors of natural selection *per se* to individual learning, but it is not clear how to proceed.

Chapter C6

Thinking about the background

C6a Context and summary

Previous chapters have presented some elements of a computational theory of routine activity and cognitive architecture. These ideas are organized around a distinction between cognitive *machinery* and the *dynamics*—that is, recurring patterns of interaction—of activity in the world. I developed these ideas about machinery and dynamics by moving back and forth between model-building and observation of actual routine activity, both through participant-observation of my own everyday activities and videotape studies of others'.

So far I have been concerned with routine activity and not with learning. I feel I understand enough about routine activity to have made a few suggestions about machinery. Despite great effort, though, I do not feel the same about about the dynamics of learning in everyday activity. This chapter describes some starting places. My goal here is not to suggest any specific machinery but rather, through my stories and my provisional analyses of them, to help focus attention on some of the phenomena.

The chapter's discussions are organized around several stories. It presents three particular stories in some detail. All three took place during a visit of several months at Oxford University. The analyses concentrate on the way in which an workspace serves as the background for its occupants' activities. Careful, but still very preliminary, consideration of this phenomenological notion of 'background' suggests ways of thinking about some of the more advanced dynamics involving deictic representation, especially some of the dynamics of learning in the course of mostly routine activities. In formulating these views I have leaned heavily on Heidegger's description of the phenomenology of everyday routine activity. My contribution is simply to help draw some helpful connections between this material and the computational theorizing that earlier chapters have begun.

Section C6b introduces the chapter's three main stories. In doing so, it distinguishes between two kinds of information about the events the stories recount. A theorist watching the events from the outside might assimilate them to larger patterns of *dynamics*

involving interactions between certain kinds of agents and certain kinds of workspaces. My own recounting of my experiences—that is, what is was like—is a matter of *phenomenology*. Dynamics and phenomenology have a subtle but close relationship, and each has a subtle but close relationship to matters of machinery.

Section C6c is a story about trying to use a computer terminal on a cluttered tabletop in the Oxford robotics lab. The story introduces the phenomenological theme of the background and its connections to the idea of deictic representation. In particular, it emphasizes that the way we experience objects in our environment is strongly informed by our current projects. This is why entities in deictic representation are individuated not just indexically but functionally as well.

Section C6d is a story about the evolution of my routines for interacting with the electric kettle in my Oxford office. This story expands on the phenomenological theme of the background by recounting how my experience of the activity of unplugging the kettle evolved over time. Entities that emerged from the background and acquired new significances retained those significances during my future interactions. This phenomenon also leads to suggestions about the dynamics of deictic representation.

Section C6e recounts several more episodes in my getting used to my Oxford office. These stories expand further on the theme of entities emerging from the background and acquiring new significances. Furthermore, the significances of objects in the room changed wholesale as new people arrived and our institutional relationship to the room evolved.

Section C6f draws lessons from these stories. It distinguishes between two ways of relating to the objects in one's surroundings, either working around them or actively changing them. When you work around something it remains very much a part of the background, mere unquestioned clutter that you have never stopped and focused on. When you actually change things, they emerge from the background and present themselves as problems to be solved. Rather than routinely and unreflectively trying to avoid them, you might ask, for example, "where would be a better place to put this?" My final suggestion is that this phenomenological distinction corresponds to an important dynamic distinction. This dynamic distinction might, I hope, lead to a more usefully concrete way of talking about learning in the context of ongoing activities.

C6b Three stories about the background

I believe that learning, by and large, occurs in the context of ongoing involvements in the world. I would like to explore this idea through three stories from my year at Oxford. All of them concern becoming accustomed to a new workspace. In particular, they all concern the way in which these newly encountered workspaces formed a background for my activity in them. All three stories take place at 19 Parks Road, Oxford. In the first story the workspace is the robotics lab, room 11; in the other two it is my office, room 27.

I took these places pretty much as they came. I had to. I did not choose them, nor

did I design them or furnish them. Nor did it ever occur to me to remake them from scratch. When I encountered them they were already arranged in accordance with the familiar patterns of laboratories and offices; I simply set about my work in them as I would in any laboratory and office. My activity in these rooms was shaped by their contents, layout, climate, and lighting, and by the habits of their other occupants. The emerging patterns of my my activity in turn left their marks: piles of papers and books, equipment stashed in and on desks, movements of chairs and doors and windows, stains from spills of tea and water, sweaters and coats and towels habitually left lying on file cabinets, and so forth. Yet with all this activity, remarkably little about the office's original layout ever got changed. Things changed when something didn't work or got in the way or was missing—or when we were neglecting our work by perversely looking around for something to change.

In telling these stories I can tell you two things, what happened and what it was like. Our goal is to fit both sorts of things into larger systems of description.

> The 'what happened' is in the realm of dynamics; that is, the recurring patterns of interaction between agents (in this case me) and environments (in this case a laboratory and an office).
>
> The 'what it was like' is in the realm of phenomenology; that is, the structure of human experience (in this case my own experience of conducting routine work in relatively unfamiliar settings).

The events recounted in these stories weren't particularly intricate, dynamically speaking, so we'll be concerned mostly with matters of phenomenology. 'Background', in particular, is a phenomenological idea. I cannot define the word in any simple way; its meaning and properties will unfold in the course of the stories.

The stories of earlier chapters have not been particularly careful in distinguishing between dynamics and phenomenology. Phenomenology is a difficult subject and this is the first chapter to attempt a systematic treatment of a phenomenological idea, namely background.

What's missing from my list, of course, is the realm of machinery; that is, whatever it was about my brain that led to my interacting with these workspaces in the ways I did. My project here is to deduce something about machinery from ideas about dynamics and phenomenology. It's always possible that vast amounts of 'unconscious processing' separate the facts of machinery from the recurrences we find in our narrations. But experience and parsimony alike have led me to believe that ideas about machinery, dynamics, and phenomenology are all tightly interconnected. In particular, talk about the background helps make ideas about 'world models' seem like a poor fit to our observations. Instead, let us focus on the ways that one takes things as they come.

C6c First story: The Oxford robotics lab

This story happened early in the year, before the robotics lab had any com-

puters to speak of. In particular, the only way to read netmail from the robotics lab was to sit at an old, rarely-used terminal in a corner which was connected to a machine across the street from which one could log in on someone else's account and use a dialup line to connect to a machine that could receive netmail. The desk on which this terminal sat was largely covered with clutter: wires, tools, electrical devices, manuals, and so forth, none of which had any particular significance for me. I was writing a paper with a friend in the US and he had netmailed me the latest draft. I didn't want to figure out how to use the local mail readers and printers, so I decided I would jot notes about the draft on a sheet of line printer paper that was left over from some earlier task and then type more elaborated versions of them in a net message to him. This story is about the clutter on the desk, but it's important that I had never been aware of remarking on the clutter or focusing on it in any way. It had simply never gotten in my way. It had remained part of the background along with details like the drawers in the desk, the cracks in the desk's veneer, the wires hanging down along the wall behind the desk, the scuffs and scratches on the terminal, and a thousand other things I never became sufficiently aware of to write down or remember. I had been sitting at this terminal exchanging netmail for a while before I started working on this paper draft; during this time I had worked around the clutter before when finding places to put things down and leave keys and pieces of paper. For example, when I wanted to put down my keys I glanced at the table near my right hand to find an unobstructed patch of brown table-top. In fact, earlier on when I wanted to make reference to the line printer paper, I had laid it on top of the clutter, which even served to tip the paper toward an angle normal to my gaze. When it came time to jot my first comment, though, I needed writing space and there wasn't any. Whereupon something neat happened. The clutter snapped out of the background into the foreground as the idea of shoving it all out of the way formed itself. I now saw the clutter *as clutter*. And I shoved it all out of the way.

In this story we encounter the notion of 'background'. A wall of the room makes a good exemplar of background: an observer composing a dynamic account would observe that the wall is always physically present and always has consequences for what one does (e.g., on account of what they prevent one from seeing). But for the purposes of one's own phenomenological narration, one only becomes 'aware' of the wall (a familiar phrase, but a vague and difficult and imprecise one) when it's time to lean on it, or bounce a ball off it, or hang something on it, or curse its failure to dampen music being overheard through it. In such situations, the wall might be said to 'emerge from the background', 'show itself', 'recommend itself' as suitable for some activity, or the like. Obviously these phrases are not meant so literally as to suggest that the wall is animate or acts of its own accord.

The whole physical setting of one's activity is also part of the background. Objects in the environment that have no significance for the currently ongoing activity stay in this background. At one point in this story a patch of table-top briefly emerged from the background to recommend itself as a place to put my keys. At another point a portion of the clutter on the desk briefly emerged from the background to recommend itself as a place to prop my listing; in this case I had only been looking for a place to lay the listing down, so the offer of a tilted 'surface' on which to rest the listing came as a pleasant—though almost subliminal—surprise. The clutter—which, after all, is only a collection of particular objects not innately labeled as 'clutter'—only emerged from the background as clutter once it became sufficiently obtrusive. It was clutter in virtue of frustrating my attempt to find a place to rest the listing when I wanted to write on it.

(This explanation illustrates one of the severe difficulties I encounter in writing these stories. In telling you about the territory in which the narrative takes place, I have to articulate for you various aspects of it that I had not had any occasion to articulate for myself as the action was actually going on. Thus, for example, I had to tell you about the 'clutter' on the desk at the beginning of the story, just because that's the simplest way to describe it, even though the story was exactly about how I came to think of the collectivity of individual objects composing it *as* 'clutter'. This problem tends to make me, as the subject of the story, sound like I understood my surroundings and equipment and materials in a much more complete and objective way than I really did. Books about fictional narration talk about this problem.)

This story teaches us some of the important features of the background.

1. An object doesn't emerge from the background 'in itself' but rather in the role it plays in the ongoing activity. The patch of desk emerges as suitable for stowing my keys, not as a naked stretch of wood or as roughly triangular in outline. (Thus it is terribly misleading, or at least critically ambiguous, to say that I 'became aware' of the patch of desk.)

2. What counts as an 'object' that emerges from the background depends on your purposes. I imagine that other people with other purposes have encountered different objects on that table than I did, and I imagine I would have encountered them myself if I had ever put together circuitry or needed a coffee-cup or wanted to look up a phone number while sitting at that desk, but I can only guess at the desk-top's actual inventory. The desk-top's materials did not appear as individual wholes whose tops happened to form a plane; instead the tops of these various materials appeared to me corporately as a roughly planar place to put my listing.

3. A given object can emerge from the background with different significances at different times. The objects that had once formed a place to rest my listing now coalesced into a larger entity that struck me as a disorganized mass of 'clutter' once 'it' got in my way.

All of these points, remember, are still about phenomenology. (In fact, Heidegger says most of this in *Being and Time* (1927), and he says it much better and in much

more detail. He doesn't use the sloppy word 'background', though I have borrowed many of his other words. Many others have said similar things, often in terms of 'figure and ground'.) Nonetheless, I believe they correspond quite closely to matters of dynamics and machinery.

Point (1), for example, must have something to do with the 'functionality' much emphasized in the theory of deictic representation. We see various objects or collections of objects assigning themselves to entities. No hyphenated gloss of these entities is going to be very accurate, but they'd be something like:

- *the-patch-of-surface-on-which-to-put-down-the-object-in-my-right-hand*
- *the-tilted-surface-on-which-to-rest-the-paper-I-intend-to-refer-to*
- *the-clutter-obstructing-my-activity-on-this-work-surface*

An object in the world, like a given patch of tabletop, might be assigned to several different entities on different occasions, according to its function (or 'role') in each occasion's ongoing activity. Indeed, an object might be assigned to several different entities at once if it is playing several roles—or, in my phenomenological terminology, if it has several significances—in the activity.

If we accept that routines involving various aspects of these entities underlay my activity in the episode I reported, then we can translate points (2) and (3) into important suggestions about deictic representation. Most particularly, entities need not be bound to individuals that would show up in any detached accounting of the 'objects' in my presence. But that's OK, since all that matters about deictic individuation of 'objects' is that all the routines pertaining to a given entity should be capable of registering their aspects and conducting their actions. As a dynamic matter, we might explain the success of a routine by observing that all the aspects it registered related to the same spatiotemporally continuous coffee mug. Other routines won't be so easily explained because their entities refer to 'objects' such as 'clutter' whose boundaries and membership happen, for dynamic reasons, never to become obtrusively uncertain.

Aside from these details, this story is actually pretty simple. In particular, it makes the 'background' sound excessively homogenous. None of the objects composing it had ever played any role in any of my past activities. Nor was any of the stuff even slightly interesting to me. Microcomputer assembler manuals bore me and I know next to nothing about electronics. Even when these objects did come into play in my activity, it was in virtue of their brute physical presence, not in terms of the uses for which they were designed. Here, then, is a story in which these matters become slightly more complicated:

C6d Second story: The electric kettle in my Oxford office

> This story takes place in my Oxford office near the end of my year there, after a fair amount of stuff had accumulated and found its place. (In particular,

this story happens after story number 3, whose details do not matter yet.) The office is about 40' long and 10' wide; one visitor compared it to a bowling lane. If you walk in the door and turn left you'll be looking down a long thin room; my desk will be centered on the far wall. To the left of my desk, against the left wall of the room, is a drafting table left over from the previous occupants. Under the drafting table are many boxes of laser-printer supplies. Along the left wall is the desk of one of my office-mates, whose name was Stephen. Between Stephen's desk and the drafting table, also along the left wall, is a table on which Stephen stores his overflow of books and papers. At the far end of this table is an electric kettle. Under the table on the wall just to the inside of its leg is a pair of electric sockets. The important point is that the area around the kettle is cramped, what with the drafting table, the boxes, Stephen's table, and a trash can I haven't mentioned yet.

As the story opens we find both electric sockets in use. The kettle is plugged into the left socket and my tape recorder is plugged into its 220V-to-110V transformer, which is plugged into the right socket. We also find the kettle sitting toward the front of the table with its handle and plug-and-switch assembly hanging off the edge. I got into the habit of setting the kettle down that way because when the kettle automatically shuts itself off as the water boils (a feature of electric kettles in this land of electrical safety fanatics) it mysteriously discharges a tablespoon of water underneath this assembly. After sweeping the resulting puddles onto the floor a few times I decided to spare myself the trouble and avoid making a mess by setting the kettle down in such a way that this water would fall on the floor (or, often, in the trash can). (I do not recall the exact history of my coming to do this routinely, else we could take apart its dynamics in more detail than this unhelpful 'because X I decided to Y'.)

And so the end of the evening came and it was time to put my tape recorder back in its desk drawer. So I squatted down by the front corner of the desk to retrieve the transformer. This has always been a clumsy matter, given the clutter of the drawing table and the trash can and boxes, not to mention the table itself. It was always necessary to adapt details of my posture to avoid all these things. But on this particular evening, as I leaned over to fetch the transformer I found the kettle poking itself in my face in an annoying fashion. So I aborted my leaning-over long enough to push the kettle back out of the way (without thinking to have any regard for my policy of hanging its switch assembly off the end of the table, not that I would have cared just then had I thought of it) and proceeded to retrieve the transformer.

The next evening as I went to retrieve the transformer, as I squatted down in the usual manner I saw the kettle in the usual place and pushed it back before reaching for the transformer at all. Later I realized that the kettle

> was not nearly so much in the way that evening as it had been the previous evening when I had first pushed it aside. I further realized that it was no more in the way on this second evening than it had been on a dozen or more other evenings, when I had taken care to avoid it no more or less than I had taken care to avoid the desk top, the near desk leg, the drafting table, the boxes, and the trash can, as part of a background of clutter.

This story exhibits many of the same features as the first one. A bunch of equipment serves as a background of clutter that I work around in my routine activities. One particular episode leads to a breakdown in which some of the equipment stands out as obtrusively in-the-way and recommends itself for being shoved away.

Some of the equipment in this story, unlike the equipment in the terminal-table story, had played a role for me in the past. I had moved Stephen's table a few inches so its leg wouldn't block access to the electric sockets. Stephen and I had moved the drafting table into that corner. Nonetheless, when I was leaning over to fetch my transformer these items had no further significance for me than obstacles for me to avoid.

Much of the equipment, moreover, has a role in other activities that routinely occupy me in that space. I regularly throw things in that trash can, usually from a few feet's distance as I sit in the chair at my desk. Indeed, when I throw something toward that trash can it often bounces off (or lands on) the boxes or the leg or support of the drafting table. When this happens, the boxes and drafting table do not stand out as 'storage' or as 'drafting table', but simply as surfaces on which my rubbish has bounced or landed.

Now consider the electric kettle. I regularly use it in making tea, an operation with many components, each of which has its own history. I mentioned my habit of replacing the kettle on the edge of the table in such a way that its leakage falls on the floor. (It was this habit of mine, recall, that was responsible for my finding the kettle in my face on the evening in question.) To take another example, when I go to refill the kettle with water, I remove its electric cord and wedge the plug in the pulley apparatus that supports the sliding straightedge on drafting table; that way it won't fall on the floor and become difficult to retrieve when I return with the refilled kettle.

Yet until the events recounted in this story, when I leaned forward to fetch my tape recorder's transformer the kettle had joined all the rest of the equipment as background—or, to be precise, as having no significance past being obstacles around which to adapt my posture.

Something about a sufficiently obtrusive obstacle makes you decide you're sick of working around it. The ongoing activity has definitely broken down, but it's not that the obstacle has absolutely prevented you from proceeding; on the evening in question I could have contorted my body one extra notch and avoided striking the kettle. Compare this story, from the sink in the robotics lab at Oxford:

> This story unfolds over several hours when I was working in the robotics lab, making myself a new cup of tea every so often as always. The whole time a mug was sitting in the sink. I vaguely found this mug familiar but didn't

> give it any thought. Part of my routine for making a new cup of tea was dumping the dregs of the previous cup into the sink. This mug was pretty much where I wanted to dump my tea each time (four times?) but I left it there, dumped around it, each time registering the minor hassle of its being slightly in the way. On one of those occasions, though, I got fed up and it recommended itself to me as needing to be moved. So I shoved it to one corner of the sink.

The two stories aren't quite analogous: the kettle was definitely more in-the-way than it had been before, whereas the mug in the sink had just been equally in-the-way on several occasions. But in each case, as in the terminal-table story, I switched—for what reason I don't exactly know—from a passive working-around to seeing the obstacle *as* an obstacle and taking action to get 'it' out of the way.

The main point is that once the kettle emerged from the background as an obstacle worthy of being pushed aside, it retained that significance on subsequent evenings. Furthermore, on those subsequent evenings the kettle emerged from the background and recommended itself for being pushed aside (a) before I even started leaning forward and (b) even though it would not have been any more in-the-way than on past occasions when I had simply adapted my posture to it as part of a generalized obstacle. This suggests that the significances of entities are not merely regenerated every time they emerge from the background. Instead, an entity's newly established significances 'stick' to it.

The kettle story recounts one episode in a trend that is always part of settling into a new space. When I first began working in my new office, I used its equipment (desk, chair, door, carpet, etc) in fairly generic ways. As time went on though, I began using the various objects in ways that were special to them. Rolling my chair about required special care on account of its sticky casters. Setting cups of tea on the desk required special care because the desktop was stained incredibly easily. (I later discovered that the stains magically disappeared with time but I never stopped taking care not to inflict them in the first place.) The kettle in particular gathered a series of new significances in the course of various problematical interactions with it: it left puddles and got in the way.

These observations suggest that a chair or desk or kettle acquires its identity through its accumulated significances. While this is not the place for a detailed treatment of this approach to the difficult question of the nature of the identity of objects, we should pause long enough to reject a couple of oversimple alternatives. One should be wary of the naive view that the world contains a collection of objectively defined individuals. While this might be true in some metaphysical sense, it begs the epistemic and representational questions of how one goes about distinguishing the supposed individuals. One should also be wary of the instrumental view that partitions an object's properties into a generic 'function' plus some additional idiosyncrasies. Manufactured commodities like electric kettles are intended to be regarded with this attitude, but the attitude itself is artificial, historically specific, and very often inappropriate.

Consider the tendency of the kettle to get in my way as I lean forward to fetch the transformer from the wall socket. This is not, narrowly speaking, a 'property of the kettle', since the kettle wouldn't necessarily get in my way in another office. Indeed, it is hard to define this 'property' at all without referring to a large network of my practices in this office: that I listen to tapes on my tape recorder as I work, that I put the tape recorder back in the desk as I go home, that there is such an institution as 'going home at the end of the day', that when using the tape recorder I set it on a nearby chair, that when I do this I plug the tape recorder into that socket, and so forth. Notice the habitual simple present tense of the verbs in these phrases and in the phrase 'the kettle gets in my way'. The habitual simple present tense indicates that the kettle's significance for me is defined in terms of a role it plays in a institutionalized routine pattern of activity that has become woven into the broader organization of my day. In general, when an object emerges from the background in the course of some activity, the 'significances' it has for that activity often (perhaps always) cannot be defined except against the activity as a whole. In this sense, the activity's very organization is itself an element of the background.

These phenomenological ideas about background and significance make important suggestions about representational issues. I have suggested that one represents objects in terms of their role in one's project. Specifically, objects are assigned to entities, which one writes with such hyphenated noun phrases as:

- *the-chair-I-am-sitting-in*
- *the-tape-recorder-I-am-listening-to*
- *the-wall-socket-the-tape-recorder-I-am-listening-to-is-plugged-into*

We have just seen, though, that the actual significances of the various objects might be impossible to specify completely. One might get started writing:

- *the-kettle-that-gets-in-my-way-as-I-lean-forward-to-unplug-...*

without any principled idea of when to stop. Fortunately, though, these hyphenated noun phrases are only our outsiders' approximate renderings of someone's actual entities. In particular, the hyphenated word-strings do not reflect structures that get traversed or rearranged by their owners. At the same time, if we cannot exhausively define these entities ourselves then we can expect their actual use to be sloppy or problematic in some way. This is not a problem with the theory, merely an element of the nature of everyday life.

C6e Third story: Moving into my Oxford office

> This is a story about my office at Oxford, the same one in which the kettle story took place. The robotics group was growing rapidly, and our office had been newly requisitioned by the department from a group studying soil

> mechanics. Stephen and I were its first occupants from the robotics group. The story recounts several episodes that took place over our first several weeks in this office.
>
> I had peeked into the office before it had been officially deeded to us. I approved of it—especially the light that came in through its two skylights—and resolved to occupy the desk along the far wall. In the event, Stephen made the first choice of a desk, moving his belongings, fortunately, into the desk along the left wall. Soon afterward, or perhaps the next day, I arrived and laid claim to the desk I had coveted. Sitting in the chair that seemed associated with my new desk, I found its casters annoyingly sticky. After failing to find a comfortable way of moving the chair around, I resolved to lay claim to one of the other chairs. I figured Stephen's chair was out of circulation, but I went around the room and tried out each of the other two chairs. (The room contained a total of five chairs, one for each of four desks and one for a table. Each of them sat on five casters and swiveled. None of them, I am sorry to say, reclined.) Neither of them was much better, but the one behind the desk nearest to the door seemed best. After extracting it from between its desk and the adjacent wall and wheeling it over to my desk, I moved my now-discarded chair to the desk whose chair I had just appropriated. In all of this I felt a certain "well, tough" selfishness that I don't particularly regret.

What's interesting throughout this story is the extent to which we took the arrangement of the room as given. We never asked ourselves how we wanted the room to be arranged, as if laying it out from scratch. For example, it wasn't until I started writing these stories in my notebook that it occurred to me to wonder how else the furniture might have been arranged. The furniture was arranged in a sufficiently customary fashion that we took it for granted that the desks were to be regarded as so many personal territories and that the various chairs were associated with their nearby desks. Indeed, it would have been odd to remark on these things at all. The room also contained two tables, one of which Stephen appropriated. The other, smaller table, the one I mentioned before, was obviously-to-all destined to house our Sun workstation. I did, however, exchange chairs with an as-yet unoccupied desk. Why? Because my original chair was a nuisance. I didn't deliberately try out this original chair to determine whether it was going to be suitable; one can rely on nuisances to bring themselves to one's attention. After appropriating my new chair, I went to the trouble of restoring the office to the customary one-to-one association of desks to chairs, for the sake of subsequent arrivals.

> The office's previous occupants were abstractions for us. We didn't even know their names. I was amused by the whole idea of 'soil mechanics' and I was fond of the archetypal soil-mechanics diagram these folks had left behind in red chalk on the office's blackboard. They had left numerous

> items behind, and I vaguely expected that they would come get them. I don't recall if they ever came for any of it, aside from some clothes stored in a file cabinet drawer.

Evidently I was so accustomed to the typical furnishing of university office spaces that I was, without being especially aware of it, regarding each of the office's objects as either 'belonging there' or 'left behind'. In my dealings with the office up to this point, the vast majority of the room's objects remained in the background. Even so, every one of them simply *had* one of these two significances.

By the way, I eventually discovered an inventory sheet attached to the inside of the office door. The office did not contain many of the items it listed, including (alas) an electric fire (the British name for an electric space heater). But, given the age of the building, I assumed it was hopelessly out of date and so didn't concern myself with the discrepancy.

> One thing they left behind was a large drafting table. Two aspects of this table soon attracted my attention. The first was a fascinating counter-weighted cable-and-pulley apparatus that permitted smooth vertical motion of the level across the drawing surface. The second was a green pad, presumably intended as a disposable backing for the drafting paper, that was attached to the table by an adhesive. This pad attracted my attention because it had partly fallen off in a peculiar manner, its adhesive having come loose at the top.

This drafting table was always oddly detached from my larger experience of the room. I'm not sure I had spent more than a couple minutes around a drafting table before, nor did this one enter into any of my routine activities. While I didn't have any clear idea of what a soil mechanic would do with a drafting table, neither did it stand out as wildly inappropriate. While each of these aspects (fascinating apparatus, pad coming loose) subsequently figured in other instructive stories, neither of them figured in any larger system of meanings in my life. In particular, neither of them led me to articulate a third aspect ...

> It took me much longer to become properly aware of a third aspect of the drafting table, viz., that it was located almost directly in front of the office's only window. It didn't actually obstruct me in walking around the office, nor did it get in my way when I walked to the window to look out at the park across the street. The window was small and one couldn't see much besides clouds out of it without walking right up to it, so I never formed any explicit notion of having my view blocked. In short, its location did not obstruct any particular activity. It merely blocked some light and made the room more cramped than it had to be. I had a growing sense of annoyance at these things, but I didn't properly articulate it until one day when I was avoiding

> my writing by looking around the office, more or less deliberately assessing it as an office. Stephen and I were both sitting at our desks. All at once I remarked to Stephen that the drafting table had to go. Stephen agreed and said that it had been bothering him as well. So I wondered aloud where we might put it. We both looked around the office for a suitable place. At length I saw that the space to the left of my desk, between my desk and the wall, was of a roughly suitable size. We cleared a channel for the drafting table—we might even have moved aside the table next to Stephen's desk—and moved the drafting table into the space. It fit quite well. Stephen asked me if I minded it there and I assured him I didn't. We both then remarked with satisfaction how much better the room looked.

As in the story of the coffee mug in the sink, the drafting table did not announce its obtrusiveness straight away. Instead it brought a cumulative, inarticulate annoyance. In the case of the drafting table, this annoyance only found expression as I looked around the room with a special and fairly artificial attitude. This attitude might be likened to the artist's method of 'looking at how it looks'. Just as few people deliberately look at the quality of the light in a scene or at the 'negative space' surrounding an object, my idly inspecting the room *qua* working space was bound up with my history of interest in architecture and semiotics.

Contrast this relatively detached attitude with the more involved, directed attitude with which Stephen and I scanned the room while looking for a space to stow the drafting table. During this search, we saw the room as so many hunks of space, each recommending itself as too small or as blocking a passage or whatever else was relevant to stowing the drafting table. The solid rectangle of space between my desk and the wall actually struck me as remarkably exact in size; in fact the drafting table fit there with comfortable precision. This was the first time we had had occasion to stow a large object in the office. But if we had had several more such occasions, I suspect that these hunks of space would have become individuals with their various particularities. They would be just as familiar as the desks and chairs.

> The anonymous soil mechanics had also left several items on the window sill. I don't recall the entire list, but they included a small potted spider plant. This plant wasn't very happy when we arrived and it proceeded to die by degrees. (I guess I don't particularly notice house plants. I am not a very good house-sitter.) There were also some sheets of line printer paper and a University telephone book. All of this stuff went untouched for many weeks. None of it got in the way of anything. I suppose I also had a vague idea that its owners might come for it. Eventually one day I resolved to throw all this stuff away. By that time we had a new office mate, named Paul. As I was pitching the scratch paper I asked them if anyone wanted this plant. Paul replied that he had adopted it so I left it alone. Last I saw it was still there.

Again the same pattern. The stuff got on my nerves. I had already resolved to get rid of it on a couple of occasions, but now I finally did it. Why did I ask if anyone else wanted the plant? I don't know for sure, but I do know that I once threw out a seemingly dead plant that someone else later proved to have been greatly attached to. (The whole psychology of plant ownership is beyond me.)

> The soil mechanics left behind a few drawers worth of aged files. I wasn't occupying any file drawers so I paid no attention to them. When Paul arrived, though, he asked Stephen and me if any of it belonged to us. Learning that it didn't he resolved to locate the soil mechanics and ask them if they wanted it. A few days later there appeared, between the door and one of the file cabinets, a few boxes of paperback books and other similar stuff. Some of the stuff was quite odd, like an album of fifty-year-old collector's cards depicting various automobiles, from a brand of German cigarettes. I assumed this stuff had been removed from the file cabinets and was awaiting disposal so I resolved to take possession of some of the more peculiar items. When I asked Stephen about the stuff, he informed me that it belonged to Paul, who had gotten it from a friend who was clearing out his attic.

As the office became more populous, it became, in a sense, more rigid. When Stephen and I had just moved in, the office's particulars all seemed up for grabs. We could switch chairs around, erase the blackboard, occupy desks, appropriate desk lamps, keep the door locked, and so forth without worrying that someone else had preferences or dibs. As the space became dense with the consequences and conventions of the settled patterns of work, every object—indeed, every aspect of the office—began referring to the others.

This condition was importantly asymmetrical. Stephen and I knew that the rubbish in the file cabinets and on the window sill had predated us. Neither we nor its presumed owners had touched it in the months we had occupied the space. Paul, however, had no way of knowing this short of asking us. (He could have inspected the stuff and found evidence of its belonging to soil mechanics. As far as I know, though, he didn't. Even if he had, for all he knew we could well have promised the soil mechanics we wouldn't throw the stuff out.) For Paul, the office as a background for his activity was saturated with the possibility that Stephen or I had established conventions that would require deference or negotiation. When Paul arrived, the terminal table, for example, was still sitting exactly where Stephen and I had found it. It was still sitting there not out of some positive decision we had made to leave it there, but rather out of its never having occurred to us to move it anywhere.

> Another instance of this effect regularly occurs on subway trains. When I board a subway car I often spot a newspaper sitting on a seat next to one of the passengers. Since I am usually desperate for reading material at such moments, it often occurs to me to grab the newspaper. The question is, does

> it belong to the person sitting next to it? Usually I don't feel like asking them. Never do I feel like just picking it up and seeing what happens. I like to think that almost all such newspapers have sat next to dozens of subway riders since being abandoned by their purchasers.

Matters of degree aside, the background is thoroughly social for everyone all the time. If I had, for instance, thought to saw a leg off a table or get a wavering fluorescent bulb replaced or carry off the carpet to my flat then the institutional context of the office would have become immediately apparent. Even the reading material on my desk acquired a social significance, insofar as I imagined that it must be giving visitors an odd impression of me. I didn't care about this and I didn't dwell on it, but it was part of the materials' significance nonetheless.

C6f The background: Phenomenology and dynamics

One of the recurring themes in these stories has been the ways in which one 'works around' the circumstances one finds. The most memorable examples were all instances of 'clutter'. I poured my tea dregs around the mug in the sink, I formed my body around the clutter of furniture near the electric socket, and I didn't insist on walking and looking through the space occupied by the drafting table. The distinguishing feature of clutter, though, is *not* that you work around it. It's clutter when it gets on your nerves and recommends itself as redundant and in-the-way.

But we also worked around many other details of the office. I never moved my desk. I never thought of changing its dimensions or its handles. It still contains the collection of pins, paper-fasteners, and other items (including a half-inch ball bearing whose origin it never once occured to me to wonder about) it contained. I never thought of rearranging its drawers, even though it occurs to me now that it'd have been mildly more convenient for the larger file-drawer to have been on the left instead of the right. And so forth forever.

We did change some things. We moved some furniture. We discarded or stowed some redundant stuff. We erased the blackboard once we wanted to use it ourselves. We got some bookshelves installed. Let us make a provisional classification of occasions for change:

> Something goes wrong or get the way or gets on our nerves. Most of my stories were about this, largely because annoyance and disaster stands out as memorable.
>
> Something is missing. Offices need bookshelves. They also need space heaters at night during the winter, but nobody would buy us one.
>
> Something is conventionally reuseable and we get around to using it, which involves clearing its past state. The chalkboard is one example. Another is the location of portable equipment. Should we even count this as change?

The background, then, manifests itself in two ways, when you're working around it and when you're changing something.

> Working-around takes things as they come. Things show themselves from the background in their role in the ongoing activity. The significance they have for your activity might depend on past experiences.
>
> Change starts with a break in the routine flow of activity. Either work is disrupted or something new has begun. One 'has something done'. Or one 'looks around' in a way that takes the environment as a resource. Or one 'puts straight' or 'puts aside' the offending things. Nonetheless the significance of the relevant materials can, and often does, make reference to the whole of the ongoing activity.

The background is a difficult idea. One is never done understanding it. Yet even my brief, coarse exposition here provides some important suggestions about the dynamics of routine activity. I have described, briefly and incompletely, some ways in which the properties of the background seem to accord with the deictic theory of representation. My larger point, though, concerns the large-scale dynamics of activity. The stories I have told about the office as the setting of work are a special case of something large and important about the world-as-a-whole as the setting of all one's activity. The two manifestations of background, in working-around and in change, correspond to two modes of activity.

> Working-around is a matter of routine: things show themselves intelligibly and in ways one can accommodate without breaking stride. No surprises. But working-around is, in another way, a matter of improvisation. One takes things as they come. Things show themselves, as it were, any way they want. One does not work from a 'world model' that explains how things will be in every detail ahead of time. This does not mean one does not learn through experience, only that the organization of activity is forever dense with the influence of the world's contingency.
>
> Changing things can be a matter of problem-solving. Something comes up. Things become complicated. One adopts much more sophisticated attitudes to things. One might try for an objective view, or read a how-to book, or lay things out on paper, or stare at the materials and imagine them going together in various ways. But no activity is any sort of pure, undiluted problem-solving. Even the most articulately abstract problem-solving is made out of individual episodes that are themselves matters of routine. And the fate of problem-solving innovations is to become folded into the larger network of improvisation and routine. Working-around is, in this sense, the prior phenomenon.

This distinction is obviously oversimple. Much more remains to be said about the phenomenology and dynamics of both routine and problem-solving. And beyond these are the hardest questions:

> Why are there these accordances between phenomenology and dynamics? And what do they mean?
>
> How does the dynamics of routine and of problem-solving constrain our cognitive machinery? And vice versa?

Finally, let us return to our original slogan:

> Learning occurs in the context of ongoing involvements in the world.

What, then, about learning? The notion of background provides a way to get started talking about 'involvement'. For one thing, one never has a complete, perfect map of the world and all the ways it bears on the practicality of one's projects. One kind of 'learning' is what's going on when problem-solving leads to changes, whether in the world or in one's routine ways of acting. Is there also a kind of learning that doesn't involve breakdowns, difficulties, annoyances, or important novelties? I think there is, but that'll have to wait. For now, at least, we can state the distinction. We can also state the question of the relationship between the two kinds of learning. If working-around is prior to changing-things, perhaps this less obtrusive and more fundamental kind of learning is the basis for all the more explicit and articulate forms. I think this is true, but I don't yet have all the words to say it.

Bibliography

Philip E. Agre, Routines, AI Memo 828, MIT Artificial Intelligence Laboratory, 1985.

Philip E. Agre and David Chapman, What are plans for?, AI Memo 1050, MIT Artificial Intelligence Laboratory, 1988. Submitted to the *AI Magazine*.

Philip E. Agre and David Chapman, Pengi: An implementation of a theory of activity, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 196-201.

James F. Allen, An interval-based representation of temporal knowledge, *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, 1981, pages 221-226.

James F. Allen, Towards a general theory of action and time, *Artificial Intelligence* **23**(2), 1984, pages 123-154.

James F. Allen and Johannes A. Koomen, Planning using a temporal world model, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, pages 741-747.

Richard Alterman, An adaptive planner, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, pages 65-69.

John R. Anderson, *The Architecture of Cognition*, Harvard University Press, 1983a.

John R. Anderson, Knowledge compilation: The general learning mechanism, *Proceedings of the International Machine Learning Workshop*, Monticello, Illinois, 1983b, pages 203-212.

Judith Bachant and John McDermott, R1 revisited: Four years in the trenches, *AI Magazine* **5**(3), 1984, pages 21-32.

Jon Barwise and John Perry, *Situations and Attitudes*, MIT Press, 1983.

John Batali, Computational introspection, AI Memo 701, MIT Artificial Intelligence Laboratory, 1983.

John Batali, A proposal for research with the goal of formulating a computational theory

of rational action, Working Paper 269, MIT Artificial Intelligence Laboratory, 1985.

Walter Benjamin, *Moscow Diary*, MIT Press, 1986. Written in German in 1926.

Eric Berne, *Games People Play*, Grove Press, 1964. Paperback from Bantam.

Eric Berne, *What Do You Say After You Say Hello?*, Grove Press, 1972. Paperback from Bantam.

Robert C. Berwick, Transformational grammar and artificial intelligence: A contemporary view, *Cognition and Brain Theory* **6**(4), 1983, pages 383-416.

Mark H. Bickhard and D. Michael Richie, On the Nature of Representation: A Case Study of James J. Gibson's Theory of Perception, Praeger, 1983.

Wilfred Bion, *Attention and Interpretation*, Tavistock, 1970.

Lawrence Birnbaum, *Integrated Processing in Planning and Understanding*, PhD Thesis, Yale University Computer Science Department, 1986. Available as Report RR-489.

Guy Blelloch, *AFL-1: A Programming Language for Massively Concurrent Computers*, Master's thesis, MIT Department of Electrical Engineering and Computer Science Department, 1986. Available as Technical Report 918, MIT Artificial Intelligence Laboratory, 1986.

Pierre Bourdieu, *Outline of a theory of practice*, Cambridge University Press, 1977.

Rodney A. Brooks, Achieving artificial intelligence through building robots, AI Memo 899, MIT Artificial Intelligence Laboratory, 1986a.

Rodney A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2**(1), April 1986b, pages 14-23.

Rodney A. Brooks and Jonathan H. Connell, Asynchronous distributed control system for a mobile robot, *Cambridge Symposium on Optical and Optoelectronic Engineering*, SPIE, October 1986, pages 77-84.

Rodney A. Brooks, Jonathan H. Connell, and Anita Flynn, A mobile robot with onboard parallel processor and large workspace arm, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, pages 1096-1100.

Rodney A. Brooks, Intelligence without representation, unpublished manuscript, 1987.

Rodney A. Brooks, Jonathan H. Connell, and Peter Ning, Herbert: A second generation mobile robot, AI Memo 1016, MIT Artificial Intelligence Laboratory, 1988.

Rodney A. Brooks, A robot that walks: Emergent behaviors from a carefully evolved network, unpublished manuscript, 1988.

Lee Brownston, Robert Farrell, Elaine Kant, and Nancy Martin, *Programming Expert*

Systems in OPS5: An Introduction to Rule-Based Programming, Addison-Wesley, 1985.

Richard L. Campbell and Mark H. Bickhard, *Knowing Levels and Developmental Stages*, S. Karger, 1986.

John F. Canny, *The Complexity of Robot Motion Planning*, MIT Press, 1987.

Jaime G. Carbonell, Derivational analogy and its role in problem solving, *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, 1983, pages 64-69.

David Chapman and Philip E. Agre, Abstract reasoning as emergent from concrete activity, in Michael P. Georgeff and Amy L. Lansky, eds, *Reasoning about Actions and Plans, Proceedings of the 1986 Workshop*, Timberline, Oregon, 1986, pages 411-424.

David Chapman, Planning for conjunctive goals, *Artificial Intelligence*, **32**(3), 1987, pages 333-377.

David Chapman, Connections, encodings, and descriptions, unpublished manuscript, 1988.

David Chapman, forthcoming PhD thesis, MIT Computer Science Department.

Raja Chatila and Jean-Paul Laumond, Position referencing and consistent world modeling for mobile robots, *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, 1985, 138-145.

R. T. Chien and S. Weissman, Planning and execution in incompletely specified environments, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 169-174.

Noam Chomsky, *Aspects of the Theory of Syntax*, MIT Press, 1965.

Jonathan H. Connell, Creature design with the subsumption architecture, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, 1987, pages 1124-1126.

Jonathan H. Connell, A behavior-based arm controller, AI Memo 1025, MIT Artificial Intelligence Laboratory, 1988.

Jonathan H. Connell, *Task-Oriented Spatial Representations for Distributed Systems*, forthcoming PhD Thesis, MIT Department of Electrical Engineering and Computer Science.

Roy Goodwin D'Andrade, The cultural part of cognition, *Cognitive Science* **5**(3), 1981, pages 179-195.

Randy Davis and Jonathan King, An overview of production systems, Memo AIM-271, Stanford Artificial Intelligence Laboratory, 1975.

Thomas L. Dean, Temporal imagery: An approach to reasoning about time for plan-

ning and problem solving, Technical Report 433, Computer Science Department, Yale University, 1985.

Thomas L. Dean and Drew V. McDermott, Temporal data base management, *Artificial Intelligence* **32**(1), 1987, pages 1-55.

Thomas L. Dean and Mark Boddy, Incremental causal reasoning, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 196-201.

Thomas L. Dean, Intractability and time-dependent planning, in Michael P. Georgeff and Amy L. Lansky, eds, *Reasoning about Actions and Plans, Proceedings of the 1986 Workshop*, Timberline, Oregon, 1986, pages 245-266.

Johan de Kleer, Jon Doyle, Guy L. Steele, Jr., and Gerald Jay Sussman, Explicit control of reasoning, *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages*, Rochester, New York, 1977. Also available as AI Memo 427, MIT Artificial Intelligence Laboratory.

Johan de Kleer, Jon Doyle, Charles Rich, Guy L Steele Jr, and Gerald Jay Sussman, AMORD: A deductive procedure system, AI Memo 435, MIT Artificial Intelligence Laboratory, 1978.

Johan de Kleer, An assumption-based TMS, *Artificial Intelligence* **28**(2), 1986, pages 127-162.

Johan de Kleer and Brian C. Williams, Reasoning about multiple faults, *Artificial Intelligence* **32**(1), 1987, pages 97-130.

Bruce R. Donald, A search algorithm for motion planning with six degrees of freedom, *Artificial Intelligence* **31**(3), 1987a, pages 295-353.

Bruce R. Donald, *Error Detection and Recovery for Robot Motion Planning with Uncertainty*, PhD Thesis, MIT Department of Electrical Engineering and Computer Science, 1987b. Available as Technical Report 982, MIT Artificial Intelligence Laboratory.

Jon Doyle, *Truth Maintenance Systems for Problem Solving*, Master's Thesis, MIT Department of Electrical Engineering and Computer Science, 1978. Also available as Technical Report 419, MIT Artificial Intelligence Laboratory, 1978.

Jon Doyle, A truth maintenance system, *Artificial Intelligence* **12**(3), 1979, pages 231-272.

Jon Doyle, *A model for deliberation, action, and introspection*, PhD Thesis, MIT Department of Electrical Engineering and Computer Science, 1980. Also available as Technical Report 581, MIT Artificial Intelligence Laboratory, 1980.

Gary Drescher, *The Schema Mechanism: A Conception of Constructivist Intelligence*, Master's Thesis, MIT Department of Electrical Engineering and Computer Science,

1985.

Hubert Dreyfus, *What Computers Can't Do*, 2nd edition, Harper and Row, 1979. The introduction to the 2nd edition is also in (Haugeland 1981), pages 161-204.

Hubert Dreyfus, *Being-in-the-world: A commentary on Heidegger's* Being and Time, *Division I*, MIT Press, forthcoming.

Edmund H. Durfee and Victor R. Lesser, Incremental planning to control a blackboard-based problem solver, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, pages 58-64.

Jeff Elman, Finding structure in time, Technical Report 8801, Center for Research in Language, University of California at San Diego, 1988.

George W. Ernst and Allen Newell, *GPS: A Case Study in Generality and Problem Solving*, Academic Press, 1969.

Michael Erdmann, *On Motion Planning with Uncertainty*, Master's Thesis, MIT Department of Electrical Engineering and Computer Science, 1987.

Michael Erdmann, Using backprojections for fine motion planning with uncertainty, *International Journal of Robotics Research* **5**(1), 1986, pages 19-45.

Gareth Evans, *The Varieties of Reference*, edited by John McDowell, Oxford University Press, 1982.

Scott E. Fahlman, A planning system for robot construction tasks, *Artificial Intelligence* **5**(1), 1974, pages 1-49.

Jerome A. Feldman, Dynamic connections in neural networks, *Biological Cybernetics* **46**(1), 1982, pages 27-39.

Jerome A. Feldman, Four frames suffice: A provisional model of vision and space, *Behavioral and Brain Sciences* **8**(2), 1985, pages 265-313.

Jerome A. Feldman and Dana H. Ballard, Connectionist models and their properties, *Cognitive Science* **6**(3), 1982, pages 205-254.

Jerome A. Feldman, Neural representation of conceptual knowledge, Technical Report 189, Computer Science Department, University of Rochester, 1986.

Richard E. Fikes, Monitored execution of robot plans produced by Strips, *Proceedings of the IFIP Congress 71*, Ljubljana, Yugoslavia, North-Holland, 1971, pages 189-194.

Richard E. Fikes and Nils J. Nilsson, Strips: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**(3), 1971, pages 189-208.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence* **3**(4), 1972, pages 251-288.

R. James Firby, An investigation into reactive planning in complex domains, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 202-206.

Jerry A. Fodor, *Representations: Philosophical Essays on the Foundations of Cognitive Science*, MIT Press, 1981.

Jerry A. Fodor, *The Modularity of Mind*, MIT Press, 1983.

Jerry A. Fodor and Zenon W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, *Cognition* **28**(1), 1988, pages 3-72.

Charles L. Forgy, OPS5 user's manual, Technical Report 81-135, Computer Science Department, Carnegie-Mellon University, 1981.

Charles L. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem, *Artificial Intelligence* **19**(1), 1982, pages 17-37.

Mark S. Fox and Stephen Smith, The role of intelligent reactive processing in production management, *13th Meeting and Technical Conference, CAM-I*, London, November 1984a.

Mark S. Fox and Stephen Smith, ISIS: A knowledge-based system for factory scheduling, *Expert Systems* **1**(1), 1984b, pages 25-49.

Harold Garfinkel, *Studies in Ethnomethodology*, Polity Press, 1984. Originally published in 1967.

Clifford Geertz, On the nature of anthropological understanding, *American Scientist* **63**(1), 1975, pages 47-53.

Dedre Gentner, Structure-mapping: A theoretical framework for analogy, *Cognitive Science*bf 7(2), 1983, pages 155-170.

Dedre Gentner and Cecile Toupin, Systematicity and surface similarity in the development of analogy, *Cognitive Science* **10**(3), 1986, 277-300.

Michael P. Georgeff, Amy L. Lansky, Reactive reasoning and planning, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 677-682.

Michael P. Georgeff, Amy L. Lansky, and Marcel J. Schoppers, Reasoning and planning in dynamic domains: An experiment with a mobile robot, Technical Note 380, SRI International Artificial Intelligence Center, 1986.

Malik Ghallab, Task execution monitoring by compiled production rules in an advanced multi-sensor robot, in Hideo Hanafusa and Hirochika Inoue, eds, *Robotics Research: The Second International Symposium*, Kyoto, 1985, pages 393-401.

James J. Gibson, *The Ecological Approach to Visual Perception*, Houghton Mifflin, 1979.

James J. Gibson, Conclusions from a century of research on sense perception, in (Koch

and Leary 1985), pages 224-230.

Georges Giralt, Raja Chatila, and Marc Vaisset, An integrated navigation and motion control system for autonomous multisensory mobile robots, in Michael Brady and Richard Paul, eds, *Proceedings of the First Symposium on Robotics Research*, MIT Press, Bretton Woods, NH, 1984, pages 191-214.

Margaret A. Hagen, James J. Gibson's ecological approach to visual perception, in (Koch and Leary 1985), pages 231-249.

Joseph Halpern, ed, *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, Monterey, California, 1986.

Kristian J. Hammond, CHEF: A model of case-based planning, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, pages 267-271.

Kristian J. Hammond, *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*, PhD Thesis, Computer Science Department, Yale University, 1986. Available as Report RR-488.

John Haugeland, ed, *Mind Design*, Bradford Books, 1981.

Patrick J. Hayes, In defence of logic, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, 1977, pages 559-565.

Philip J. Hayes, A representation for robot plans, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 181-188.

Barbara Hayes-Roth and Frederick Hayes-Roth, A cognitive model of planning, *Cognitive Science* **3**(4), 1979, pages 275-310.

Martin Heidegger, *Being and Time*, translated by John Macquarrie and Edward Robinson, Harper and Row, 1961. Originally published in German in 1927.

John Heritage, *Garfinkel and Ethnomethodology*, Polity Press, 1984.

R. D. Hinshelwood, *What Happens in Groups*, Free Association Books, 1987.

Geoffrey E. Hinton and James A. Anderson, *Parallel Models of Associative Memory*, Lawrence Erlbaum Associates, 1981.

Geoffrey E. Hinton, Implementing semantic networks in parallel hardware, Chapter 6 in (Hinton and Anderson 1981), pages 161-187.

Geoffrey E. Hinton and David S. Touretzky, Symbols among the neurons: Details of a connectionist inference architecture, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, 1985, pages 238-243.

Geoffrey E. Hinton, James L. McClelland, and David E. Rumelhart, Distributed representation, Chapter 3 in (Rumelhart and McLelland 1986), pages 77-109.

John H. Holland, *Adaptation in Natural and Artificial Systems*, Michigan University Press, 1975.

Ian D. Horswill, *Reactive Navigation for Mobile Robots*, Master's thesis, MIT Department of Electical Engineering and Computer Science, 1988.

Ian D. Horswill and Rodney A. Brooks, Situated vision in a dynamic world: Chasing objects, *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, 1988, pages 796-800.

Edwin Hutchins, Mediation and automatization, ICS Report 8704, Institute for Cognitive Science, University of California at San Diego, 1987.

Margaret Jean Intons-Peterson and JoAnne Fournier, External and internal memory aids: When and how often do we use them?, *Journal of Experimental Psychology: General* **115**(3), 1986, pages 267-280.

Philip N. Johnson-Laird and Peter C. Wason, *Thinking: Readings in Cognitive Science*, Cambridge University Press, 1977.

Michael I. Jordan, *Serial order: A parallel, distributed approach*, Report 8604, Institute for Cognitive Science, University of California at San Diego, 1986.

Michael I. Jordan and David A. Rosenbaum, Action, in Michael I. Posner, ed, *Handbook of Cognitive Science*, MIT Press, forthcoming.

Leslie Pack Kaelbling, Rex: A symbolic language for the design and parallel implementation of embedded systems, *Proceedings of the AIAA Conference on Computers in Aerospace*, Wakefield, Massachusetts, 1987.

Leslie Pack Kaelbling, Goals as parallel program specifications, *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, 1988, pages 60-65.

Kenneth Kaye, *The Mental and Social Life of Babies: How Parents Create Persons*, University of Chicago Press, 1982.

Raymond Klein, Inhibitory tagging system facilitates visual search, *Nature* **334**, 4 August 1988, pages 430-431.

Donald Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, 1983.

Sigmund Koch and David E. Leary, *A Century of Psychology as Science*, McGraw-Hill, 1985.

Janet L. Kolodner and Richard E. Cullingford, Towards a memory architecture that supports reminding, Report 86-10, School of Information and Computer Science, Georgia Institute of Technology, 1986.

Robert A. Kowalski, Predicate logic as a programming language, *Proceedings of the IFIP Congress 74*, North-Holland, 1974, pages 569-574.

John E. Laird, *Universal Subgoaling*, PhD Thesis, Computer Science Department, Carnegie-Mellon University, 1983. Available in (Laird, Rosenbloom, and Newell 1986).

John E. Laird and Allen Newell, A universal weak method: Summary of results, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, pages 771-773.

John E. Laird, Paul S. Rosenbloom, and Allen Newell, Toward chunking as a general learning mechanism, *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, 1984, pages 188-192.

John E. Laird, Paul S. Rosenbloom, and Allen Newell, Chunking in Soar: The anatomy of a general learning mechanism, *Machine Learning* **1**(1), January 1986, pages 11-46.

John E. Laird, Paul S. Rosenbloom, and Allen Newell, *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Kluwer, 1986.

John E. Laird, Allen Newell, and Paul S. Rosenbloom, Soar: An architecture for general intelligence, *Artificial Intelligence* **33**(1), 1987, pages 1-64.

John E. Laird, Recovery from incorrect knowledge in Soar, *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, 1988, pages 618-623.

Amy L. Lansky and David S. Fogelsong, Localized representation and planning methods for parallel domains, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 240-245.

Jill Larkin and Herbert A. Simon, Why a diagram is (sometimes) worth ten thousand words, *Cognitive Science* **11**(1), 1987, pages 65-100.

Bruno Latour, Sociology of a door, *Twente II*, Eschende, Holland, 1987.

Jean Lave, *Cognition in Practice*, Cambridge University Press, 1988.

Douglas B. Lenat, The nature of heuristics, *Artificial Intelligence* **19**(2), 1982, pages 189-249.

Douglas B. Lenat, Eurisko: A program that learns new heuristics and domain concepts, *Artificial Intelligence* **21**(1), 1983, 61-98.

Douglas B. Lenat and John Seely Brown, Why AM and Eurisko appear to work, *Artificial Intelligence* **23**(3), 1984, pages 269-294. A shorter version appears in *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, 1983, pages 236-240.

Hector J. Levesque and Ronald J. Brachman, Expressiveness and tractability in knowl-

edge representation and reasoning, *Computational Intelligence* **3**(2), 1987, pages 78-93.

Theodore Linden, Plan and goal representations, in William Swartout, ed, DARPA Santa Cruz Workshop on Planning, *AI Magazine* **9**(2), 1988, pages 115-131.

Tomás Lozano-Pérez, Matthew T. Mason, and Russell H. Taylor, Automatic synthesis of fine-motion strategies for robots, *International Journal of Robotics Research* **3**(1), 1984, pages 3-24.

Kevin Lynch, *The Image of the City*, MIT Press, 1960.

James V. Mahoney, *Image Chunking: Defining Spatial Building Blocks for Scene Analysis*, Master's Thesis, MIT Department of Electrical Engineering and Computer Science, 1987.

Mitchell Marcus, *A Theory of Syntactic Recognition for Natural Language*, MIT Press, 1980.

David Marr, A theory for cerebral neocortex, *Proceedings of the Royal Society of London B* **176**, pages 161-234, 1970.

David Marr, Artificial intelligence: A personal view, *Artificial Intelligence* **9**(1), 1977, pages 37-48.

David Marr, *Vision*, Freeman, 1982.

David McAllester, *ONTIC: A Knowledge Representation System for Mathematics*, MIT Press, 1988.

James L. McClelland and David E. Rumelhart, An interactive activation model of the effect of context on language learning (Part I), *Psychological Review* **88**(5), 1981, pages 375-407.

James L. McClelland, Putting knowledge in its place: A scheme for programming parallel processing structures on the fly, *Cognitive Science* **9**(1), 1984, pages 113-146.

Drew V. McDermott, *Flexibility and Efficiency in a Computer Program for Designing Circuits*, Technical Report 402, MIT Artificial Intelligence Laboratory, 1977.

Drew V. McDermott, Planning and acting, *Cognitive Science*, **2**(2), 1978, pages 71-109.

Drew V. McDermott, A temporal logic for reasoning about processes and plans, *Cognitive Science* **6**(2), 1982, pages 101-155.

Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, eds, *Machine Learning: An Artificial Intelligence Approach*, Tioga, 1983. Reprinted by Morgan Kaufmann, 1983.

Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, eds, *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, 1986.

George A. Miller, Eugene Galanter, and Karl H. Pribram, *Plans and the Structure of Behavior*, Henry Holt and Company, 1960.

Marvin Minsky, A framework for representing knowledge, AI Memo 306, MIT Artificial Intelligence Laboratory, June 1974. Also in part in (Winston 1975a) pages 211-277 and in (Haugeland 1981) pages 95-128. Another version appears in Proc TINLAP 1975 and in (Johnson-Laird and Wason 1977) pages 355-376.

Marvin Minsky, K-Lines: A theory of memory, *Cognitive Science* 4(2), 1980, pages 117-133.

Marvin Minsky, Jokes and the logic of the cognitive unconscious, AI Memo 603, MIT Artificial Intelligence Laboratory, 1980.

Marvin Minsky, *The Society of Mind*, Simon and Schuster, 1986.

Tom Mitchell, Learning and problem solving, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, pages 1139-1151.

Raymond Mooney and Gerald DeJong, Learning schemata for natural language processing, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, 1985, pages 681-687.

John H. Munson, Robot planning, execution, and monitoring in an uncertain environment, in *Proceedings of the Second International Joint Conference on Artificial Intelligence*, London, 1971, pages 338-349.

Robert Neches, *Models of Heuristic Procedure Modification*, PhD Thesis, Psychology Department, Carnegie-Mellon University, 1981.

Allen Newell, J. Clifford Shaw, and Herbert A. Simon, Report on a general problem solving program, in *Proceedings of the International Conference on Information Processing*, UNESCO, Paris, 1960, pages 256-264.

Allen Newell and Herbert A. Simon, GPS: A program that simulates human thought, in Edward A. Feigenbaum and Julian Feldman, eds, *Computers and Thought*, McGraw-Hill, 1963, pages 279-296.

Allen Newell and Herbert A. Simon, *Human Problem Solving*, Prentice-Hall, 1972.

Allen Newell, Production systems: Models of control structures, in W. G. Chase, ed, *Visual Information Processing*, Academic Press, New York, 1973.

Milind Tambe and Allen Newell, Some chunks are expensive, *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, Michigan, 1988, pages 451-458.

Nils J. Nilsson, *Principles of Artificial Intelligence*, Tioga, 1980.

Nils J. Nilsson, Shakey the Robot, Technical Note 323, SRI Artificial Intelligence Center, 1984.

Donald A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988.

Elizabeth F. Preston, *Representational and Non-Representational Intentionality: Husserl, Heidegger, and Artificial Intelligence*, PhD Thesis, Department of Philosophy, Boston University, 1988.

Gregory Provan, Efficiency analysis of multiple-context TMSs in scene representation, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 173-177.

Gregory Provan, *Complexity Analysis of Truth Maintenance Systems, with Application to High Level Vision*, DPhil thesis, Department of Engineering, University of Oxford, forthcoming.

R. Bruce Roberts and Ira P. Goldstein, The FRL manual, AI Memo 409, MIT Artificial Intelligence Laboratory, June 1977.

Barbara Rogoff and Jean Lave, eds, *Everyday Cognition: Its Development in Social Context*, Harvard University Press, 1984.

Richard Rorty, *Philosophy and the Mirror of Nature*, Princeton University Press, 1979.

Paul S. Rosenbloom, *The Chunking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility*, PhD Thesis, Department of Computer Science, Carnegie-Mellon University, August 1983.

Stanley J. Rosenschein, Formal theories of knowledge in AI and robotics, *New Generation Computing* **3**(4), pages 345-357.

Stanley J. Rosenschein and Leslie Pack Kaelbling, The synthesis of digital machines with provable epistemic properties, in (Halpern 1986), pages 83-98.

David E. Rumelhart and James L. McLelland, eds, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, MIT Press, 1986.

David E. Rumelhart, Geoffrey E. Hinton, and R. J. Williams, Learning internal representations by error propagation, Chapter 8 in (Rumelhart and McClelland 1986), volume 1, pages 318-362.

David E. Rumelhart, Paul Smolensky, James L. McClelland, and Geoffrey E. Hinton, Schemata and sequential thought processes in PDP models, Chapter 14 in (Rumelhart and McClelland 1986), volume 2, pages 7-57.

Earl D. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* **5**(2), 1974, pages 115-135.

Earl D. Sacerdoti, The nonlinear nature of plans, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 206-214.

Earl D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier, 1977.

Harvey Sacks, Emanuel A. Schegloff, and Gail Jefferson, A simplest systematics for the organization of turn-taking in conversation, in Jim Schenkein, ed, *Studies in the Organization of Conversational Interaction*, Academic Press, 1978, pages 7-55.

Roger C. Schank and Robert P. Abelson, Scripts, plans, and knowledge, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 151-157.

Roger C. Schank and Robert P. Abelson, *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, 1977.

Roger C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge University Press, 1982.

Bob Scher, *The Fear of Cooking*, Houghton-Mifflin, 1984.

Marcel J. Schoppers, Universal plans for reactive robots in unpredictable environments, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, 1987, pages 1039-1046.

Alfred Schutz, *The Phenomenology of the Social World*, translated by George Walsh and Frederick Lehnert, Heinemann, 1967. Originally published in German in 1932.

Sylvia Scribner, Studying working intelligence, in (Rogoff and Lave 1984), pages 1-40.

Yoav Shoham, Temporal logics in AI: Semantical and ontological considerations, *Artificial Intelligence* **33**(1), 1987, pages 89-104.

Yoav Shoham, *Reasoning About Change*, MIT Press, 1988.

John Shore, *The Sachertorte Algorithm and Other Antidotes to Computer Anxiety*, Viking, 1975.

Reid G. Simmons, *Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems*, PhD Thesis, MIT Department of Electrical Engineering and Computer Science, 1988a.

Reid G. Simmons, A theory of debugging plans and interpretations, *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, 1988b, pages 94-99.

Herbert A. Simon, *The Sciences of the Artificial*, MIT Press, 1970.

Dorothy G. Singer and Tracey A. Revenson, *A Piaget Primer: How a Child Thinks*, New American Library, 1978.

Steven Small, Viewing word expert parsing as linguistic theory, *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Vancouver, 1981, pages 70-76.

Steven Small, Garrison Cottrell, and Lokendra Shastri, Toward connectionist parsing, *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, 1982, pages 247-250.

Brian Cantwell Smith, Varieties of self-reference, in (Halpern 1986a), pages 19-44.

Brian Cantwell Smith, The correspondence continuum, *Proceedings of the 6th Canadian AI Conference*, Montreal, 1986b.

Richard M. Stallman and Gerald Jay Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence* **9**(2), 1977, pages 135-196.

Mark Stefik, Planning and meta-planning, *Artificial Intelligence* **12**(2), 1981, pages 141-170.

Mark Stefik, *Planning with Constraints*, PhD Thesis, Computer Science Department, Stanford University, 1980. Also Stanford Computer Science Department Memo 80-784.

Lucy Suchman, What is a plan?, ISL Technical Note, Xerox Palo Alto Research Center, 1986.

Lucy Suchman, *Plans and Situated Action*, Cambridge University Press, 1987.

Harry Stack Sullivan, *The interpersonal theory of psychiatry*, Norton, 1953.

Gerald Jay Sussman and Drew McDermott, Why conniving is better than planning, MIT AI Lab Memo 255A, 1972, also in *Proceedings of the Fall Joint Computer Conference* **41**, AFIPS Press 1972, pages 1171-1179.

Gerald Jay Sussman, *A Computer Model of Skill Acquisition*, Elsevier, 1975.

Robert Tarjan, Algorithm design, *Communications of the ACM* **30**(3), 1987, pages 205-212.

Austin Tate, Interacting goals and their use, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 215-218.

Austin Tate, Planning and Condition Monitoring in a FMS, *International Conference on Flexible Manufacturing Systems*, London, July 1984.

Austin Tate, A review of knowledge-based planning techniques, *The Knowledge Engineering Review* **1**(3), June 1985, pages 4-17.

David S. Touretzky, A distributed connectionist production system, Technical Report 86-172, Computer Science Department, Carnegie-Mellon University, 1986.

Ann Treisman and Garry Gelade, A feature integration theory of attention, *Cognitive Psychology* **12**(1), 1980, pages 97-136.

Shimon Ullman, Against direct perception, *Behavioral and Brain Sciences* **3**(3), 1980, pages 373-381. With peer commentary on pages 381-408 and author's response on pages 408-415.

Shimon Ullman, Visual routines, *Cognition* **18**, 1984, pages 97-159.

Steven A. Vere, Planning in time: Windows and durations for activities and goals, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**(3), 1983, pages 246-267.

Paul Viola, *Mobile Robot Evolution*, Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, 1988.

Lev Vygotsky, *Thought and Language*, translated and edited by Alex Kozulin, MIT Press, 1986. Originally published in Russian in 1934.

Lev Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*, edited by Michael Cole, Vera John-Steiner, Sylvia Scribner, and Ellen Souberman, Harvard University Press, 1978.

Richard Waldinger, Achieving several goals simultaneously, Technical Note 107, SRI Artificial Intelligence Center, 1975.

David Warren, Warplan: A system for generating plans, Memo No. 76, Department of Computational Logic, University of Edinburgh, 1974.

David Warren, An abstract Prolog instruction set, Technical Note 309, SRI International AI Center, 1983.

Donald A Waterman and Frederick Hayes-Roth, eds, *Pattern-Directed Inference Systems*, Academic Press, 1978.

James W. Wertsch, *Vygotsky and the Social Formation of Mind*, Harvard University Press, 1985.

Robert Wilensky, Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding, *Cognitive Science* **5**(3), 1981, pages 197-233.

Robert Wilensky, *Planning and Understanding: A Computational Approach to Human Reasoning*, Addison-Wesley, 1983.

David E. Wilkins, Representation in a domain-independent planner, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, pages 733-740.

David E. Wilkins, Domain-independent planning: Representation and plan generation,

Artificial Intelligence **22**(3), 1984, pages 269-301.

David E. Wilkins, Recovering from execution errors in SIPE, *Computational Intelligence* **1**(1), 1985, pages 33-45.

David E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, 1988.

David Willshaw, Holography, associative memory, and inductive generalization, Chapter 3 in (Hinton and Anderson 1981), pages 83-104.

Donald Winnicott, *Playing and Reality*, Tavistock, 1971.

Donald Winnicott, *On Human Nature*, Basic Books, 1988.

Patrick H. Winston, *The Psychology of Computer Vision*, McGraw-Hill, 1975.

Patrick H. Winston, Learning and reasoning by analogy, *Communications of the ACM* **23**(12), 1979, pages 689-703.

Patrick H. Winston, Learning by augmenting rules and accumulating censors, *Proceedings of the International Machine Learning Workshop*, Monticello, Illinois, 1983, pages 2-11. Also Chapter 3 in (Michalski, Carbonell, and Mitchell 1986), pages 46-61.

Patrick H. Winston, *Artificial Intelligence*, 2nd edition, Addison-Wesley, 1984.