Execution architectures and compilation

Stuart J. Russell*
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

This paper introduces a partition of the possible forms of knowledge according to their relationship to the basic objective of an intelligent agent, namely to act successfully in response to its environment. The resulting classes of knowledge range from fully declarative to fully compiled. From these classes, it is possible to generate 1) a set of *execution architectures,* each of which combines some of the classes to produce decisions; and 2) a set of *compilation methods,* that transform knowledge into more efficient but (approximately) behaviourally equivalent, compiled forms. Existing compilation methods can be understood within this framework, and new compilation methods and execution architectures are indicated. It is proposed that systems with the ability to learn, use and transform between all the types of knowledge may be able to achieve simultaneously higher levels of competence, efficiency and flexibility.

1 Introduction

Artificial intelligence in general, and machine learning in particular, must find a route from raw perceptions of the world to successful actions within that world. In this paper, I try to identify a few more way-stations on this daunting route. In doing so in so small a space I will need to ride roughshod over some important questions, but the basic thesis is that there are several distinct *kinds* of knowledge that can be acquired from perceptions and used for acting, and therefore a multiplicity of execution architectures that effect the input/output mapping for an agent. The approach also helps one to understand the variety of existing execution architectures by pointing out the possibilities for compilation transformations that preserve behavioural equivalence.

The paper begins by motivating compilation as a useful notion in AI. In order to find out what kinds of compilation there are, I start from an uncompiled architecture and work down. To do this, I first describe what I mean by autonomy and hence what might be an uncompiled architecture for an autonomous intelligent agent. I then discuss the possible forms of knowledge, from fully declarative to fully compiled, and map out the compile tion paths linking them. Finally, I discuss the resulting research agenda and speculate on why declarative knowledge is worth having.

*This work was supported by the Lockheed AI Center and the State of California MICRO program

2 Why compilation?

It is a commonplace of artificial intelligence that perfect rationality, in the sense prescribed by decision theory, is unlikely to be computationally attainable by systems that explicitly solve the decision problem at each juncture. Simon [1982b] made clear the distinction between systems that compute the rational thing to do (procedural rationality), and systems that simply do the rational thing (substantive rationality). Systems whose execution architectures are based on explicit use of declarative knowledge to reach decisions to act seem to suffer from a good deal of overhead, both in terms of time and extra cognitive machinery. A currently popular notion is that all this deliberation is a waste of time — why don't we just build agents that "do the right thing" [Brooks, 1986, Agre and Chapman, 1987]? Substantive rationality, however, does not come for free. Although it means that an agent can be perfectly rational despite limited computational resources, it can only arise in one of three ways:

1. By design, where the *designer* possesses the computational and informational resources required to find optimal solutions.
2. By simple adaptation, that is, direct adjustment of behaviour in response to feedback from the environment.
3. By deliberative self-design, where the agent itself carries out the required computations, (perhaps incrementally) compiling them to ensure substantive rationality in future situations.

Many people in AI are interested in ultimately creating *autonomous* intelligent systems. Below, we develop a simple notion of autonomy that makes the first of these three options less than desirable. At the conclusion of this paper, we offer speculation as to why the second option may also be inappropriate. The majority of researchers, including many advocates of substantive rationality, believe that the most promising route to achieving intelligence lies in systems capable of acquiring and using knowledge in a declarative form, and gradually compiling it for use in more efficient execution architectures.

There seems little doubt that such efficient execution architectures do exist. A clock does the right thing as a direct result of its fixed structure, with no significant perceptive ability. A pianola or 'player-piano' executes a behaviour by directly interpreting a stored sequence of action descriptions. State-free, feed-forward networks can implement more complex mappings from inputs to actions; Agre and Chapman [1987] have advocated such systems as a reasonable architecture for intelligent systems. Connectionist systems follow a similar design philosophy. All of these approaches to producing behaviour have significant advantages in terms of simplicity and computation time. In a sense, they all implement 'condition-action' rules, or *productions,* with the limitation that conditions must be computable directly from current sensory inputs. If we design systems using more declarative constructs, the performance demands of real environments will necessitate some mechanism for converting inefficient but general decision-making methods into a form that displays greater alacrity.[1]

Essentially, compilation is a method for omitting intermediate computations in the input-output mapping. Computations can be omitted when their answers are already known, so that subsequent computations can be modified to use those answers directly rather than having them recomputed first. Compilation is only useful when an entire *class* of computations can be omitted, so that a whole class of decision-making episodes can be speeded up. Therefore another view of compilation is as a means for taking advantage of *regularities* in the environment. For example, I have learnt, when driving to work, to turn left at the bright orange Oscar's Burgers sign. This is because the sign is always at the same street, and that street always leads to the parking lot; but this information is now only implicit in my performance.

Researchers have found ways to add some form of compilation into whatever system they use as a performance element. The most usual forms of compilation involve collapsing operator sequences and collapsing implications in a logical system. Anderson [1986] developed *knowledge compilation* to speed up a production system. Rosenbloom [1987] developed *chunking* to compile the impasse resolution procedures in SOAR , also a production system. Fikes and Nilsson [1972] developed the triangle-table method to form *macro-operators* to speed up problem-solving in STRIPS. *Explanation-based learning* [Mitchell *et* a/., 1986] compiles simple forms of rule-based inference. To date, the technique has only been applied to concept membership problems and what might be called 'existential' problem-solving, in which *any* action that eventually leads to a solution is acceptable.

It is possible to unify all these techniques as points in a well-defined space of possible compilation methods. The route we will take is to analyze the possible general classes of compiled knowledge, and then generate a space of compilation methods as routes for converting between and within these various knowledge classes. In order to do this, some notion of an *uncompiled formulation* for autonomous decision-making is needed. To motivate the choice of a decision-theoretic framework to fill this role, a brief digression on autonomous agents is in order.

[1]Subramanian and Woodfill [1989] have given an example of how a situation-calculus-based planner might generate propositional condition-action rules of the form used by Agre and Chapman.

3 Autonomy

A system is autonomous to the extent that its behaviour is determined by its immediate inputs and past experience, rather than by its designer's. A system that operates on the basis of built-in assumptions will only operate successfully when those assumptions hold, and thus lacks flexibility. A truly autonomous system should be able to operate successfully in any universe, given sufficient time to adapt. The system's internal knowledge structures should therefore be constructive, in principle, from its experience of the world.[2]

For a notion of learning in such systems, the following definition seems acceptable: learning takes place when the system makes changes to its internal structure so as to improve some metric on its long-term future performance, as measured by a fixed performance standard (cf. Simon's [1983] definition). It also seems clear that the performance standard must ultimately be externally imposed [Buchanan *et* a/., 1979], particularly since, for the purposes of building useful artifacts, modification of the performance standard to flatter one's behaviour does not exactly fit the bill.

There are three[3] essential aspects of experience:

1. Perceptions that reflect the current state of the environment.
2. Perception of the agent's own actions.
3. Information as to the quality of the agent's performance.

The agent's perceptions may be partial, intermittent and unreliable. The *truth* of the agent's perceptions is irrelevant (or, to put it another way, each perception carries a guarantee of its own truth). What is important is that the perceptions be *faithful* in the following sense: there is a consistent relationship between the agent's perceptions and the performance feedback. The relationship can be arbitrarily complex and uncertain — the more so, the more difficult the learning problem.

Given these basic categories of inputs, some obvious candidates for the constituents of the uncompiled architecture would include beliefs about the state of the world, beliefs about the effects of actions and beliefs about the relationship between the state of the world and the level of performance quality feedback. Each of these can be 'explained' only by appealing to the agent's direct experience of the world or to prior knowledge of the same type, rather than being derivable from other knowledge structures (this point can be argued in a good deal more detail). An appropriate uncompiled architecture for an intelligent system would therefore seem to be decision-theoretic in nature.

The strongest alternative to this proposal is the *goal-based* architecture (as proposed by, for example,

[2]1 don't wish to equate autonomous systems with *tabula rasa* systems, however, since this seems a somewhat impractical way to proceed. A reasonable halfway-point is to design systems whose behaviour is determined in large part, at least initially, by the designer's knowledge of the world, but where all such assumptions are as far as possible made explicit and amenable to change by the agent. This sense of autonomy seems also to fit in reasonably well with our intuitive notions of intelligence.

[3]One might argue that perception of the agent's internal computations is also necessary for certain kinds of learning; these can be included in the 'environment' and 'actions'.

Newell [1981] and the SOAR group [Laird *et al.*, 1987]). In such systems, the idea of a utility measure is replaced by the idea of a goal — an intensional description of a class of desirable states. It might be argued that goals and utility functions are equally valid alternative formulations; for example, one could construct a utility function from a goal by assigning high utility to goal states and lower utility to other states; and one could identify goals with classes of high-utility states from a given utility function. However, there are two significant objections to such a proposal. First, goal-based formalisms have a hard time dealing appropriately with conflicting goals, and thus cannot adequately model the desires of complex agents (who, for example, may want a Jaguar *and* a new roof, but cannot afford both). Utility functions can easily deal with such cases. The second, and perhaps more important, objection involves the nature of the performance feedback that an agent receives. It seems to me that this feedback must be essentially *non-representational* The agent is not given goal descriptions or utility functions by the environment, only a series of point values from the external performance metric. In this way, the environment does not need to know the agent's representation scheme in order to 'train' it. There merely has to be agreement on what counts as 'warm' and what counts as 'cold'. In the case of evolution, the metric 'more offspring is better' is self-defining. In the RALPH Rational Agents with Limited Performance Hardware project at Berkeley, the agents operate in a real-time, simulated environment where the performance feedback reflects the amount of food the agents find and consume, and their successful avoidance of enemies who might wound them. Each ralph is designed to induce a utility function from the performance feedback data. However, this utility function will not, in general, reproduce the function that generates the performance signal. Instead, it should converge to a function that *predicts* the long-term expectation of the performance signal, given the current state. In this way, the agent can use a simple, 'greedy' decision procedure that avoids extensive lookahead. One would expect the utility function to be much more complex than the performance signal generator.

4 Categories of knowledge

To recap: the basic categories of knowledge in an uncompiled system, that is, a system operating with a decision-theoretic formulation, are

- knowledge about the state of the world (this includes direct perceptions, and rules relating parts of the world state, such as 'if it is raining, the ground is usually wet');
- knowledge about the results of actions, i.e., constraints on the world state *after* an action has been performed;
- knowledge about the (relative) utility of a world state.

Decisions are made by selecting the action that results in the next state of highest expected utility.

We will use an informal[4] notation as follows:

[4] Readers interested in a more thorough and formal development of declarative formulations of agents are referred to Doyle's recent work [Doyle, 1988].

$$\begin{tikzcd}
condition(s) \arrow[loop above, "A"] \arrow[d, "B"] \arrow[dd, bend left=50, "E"] \arrow[ddd, bend left=70, "D"] \\
condition(result(a,s)) \arrow[d, "C"] \arrow[dd, bend right=60, "F"'] \\
utility(result(a,s),v) \arrow[d, "DT"] \\
best(a,s)
\end{tikzcd}$$

Figure 1: Forms of compiled and uncompiled knowledge

- *condition(state)* represents an arbitrary predication on a world state;
- *utility(state, value)* represents an arbitrary predication about the absolute or relative utility of a world state;
- *result (action,state)* represents an arbitrary predication about the resulting state after taking an action in a given state;
- *best (action,state)* means that *action* is the best available in the given state.
- *CurrentState* refer to the state of the world in which the agent finds itself.

A basic decision procedure derives conclusions of the form *utility(result(action,CurrentState),value)* for each available action, and uses the decision-theoretic principle to conclude *best (action, CurrentState)* for one of them. In order to obtain these utilities, then, it will normally need to know some *condition(CurrentState),* allowing it to conclude *condition(result(action, CurrentState)),* from which it can infer the utility of taking the action. The four stages of 'static' knowledge are shown in Figure 1, with the obvious abbreviations.

In the same notation, the forms of 'dynamic' knowledge needed to link these stages together are as follows:

A: *condition(state)* => *condition(state)*

B: *condition(state)* => *condition (result(action, state))*

C: *condition(state)* => *utility (state, value)*

In addition, the decision-theoretic principle (labelled DT in the diagram) takes knowledge of the utility of actions and concludes that one is best.

The principle of compilation is to convert a formulation in which each of these kinds of knowledge is used explicitly into a formulation that results in the same decisions but is computationally more efficient. By examining the figure we can see the additional kinds of knowledge that can offer shortcuts for the decision procedure. They are as follows:

- D: *condition(state)* => *best(action, state)*
- E: *condition(state)* => *utility(result(action, state), value)*
- F: *condition(result(action, state))* => *best(action, state)*

Type D rules are the standard *condition-action rules* used in production systems; for example, "IF a car is coming straight for you THEN jump out of the way". Such rules compile away any knowledge of the results of the action or of the reasons for those results' desirability.

Type E rules could be called *action-utility rules;* for example, the value of a forking move in chess is typically the difference between the value of the lesser of the forked pieces and the value of the forking piece. Such rules avoid explicit computation of the action's results.

Type F rules are extremely interesting. Consider the case of a type F rule with universal quantification over the action and state arguments:

$$\forall a, s[condition(result(a, s)) \Rightarrow best(a, s)].$$

Essentially, such a rule says that an action should be executed whenever the situation is such that the action will achieve a certain condition. An agent using such a rule thus believes the condition to be desirable independent of the side effects of its achievement on the rest of the agent's utility function. This is exactly the definition of a *goal* that is used in Newell's knowledge-level agent architecture [Newell, 1981]. Goals are therefore compiled from a decision-theoretic formulation, when the agent believes that a condition is 'separately optimizable'. Since goals allow backward-chaining rather than forward-chaining for selecting actions, they can provide huge efficiency gains, and their creation by compilation is an important, unstudied process.

In this context, an *execution architecture* is an interpreter that uses some combination of facts of various types to reach a decision. Three basic architectures that can be implemented using knowledge of types A through F are as follows:

1. Decision-theoretic systems: Knowledge of types A, B and C is combined to find the best action using the DT principle.
2. Production systems: Knowledge of type D provides action choices directly.
3. Goal-based systems: Knowledge of types A, B and F suggests actions that achieve the desired goal condition.

Systems based on combining knowledge of type E with the DT principle seem not to have been studied systematically. Such systems organize their utility knowledge around actions rather than states. They are discussed further below.

5 The space of compilation methods

The space of compilation methods can now be generated by looking at various ways in which all these forms of knowledge can be combined to produce more operational versions of the same underlying theory. There are two basic classes of compilation methods: *homogeneous* and *heterogeneous.*

5.1 Homogeneous compilation

The first two forms of uncompiled knowledge (A and B) have left and right-hand sides of the same form, and therefore allow indefinite chaining of inferences. The chained inferences can be compiled to make the calculation of the requisite basic forms of declarative knowledge more efficient:

- $A + A \longrightarrow A$
- $B + B \longrightarrow B$

These two compilation modes are already well-known in the literature. *Explanation-based learning* is usually used to compress chains of inferences about the state of the world. For example, the first time a particular type of bridge is designed, very long computations are needed to predict its safe load from its structural description; the results are then saved as a rule about this class of bridges. *Macro-operator formation* compresses inferences about the results of sequences of actions. After map-tracing and trial and error, I discover a good route to work; then I compile it into an automatic routine, or action sequence, to get me there.

The reason for the popularity of these forms of compilation is obvious: a uniform architecture (one based on just type A or just type B knowledge) is closed under these compilation methods. In other words, the same execution architecture applies to the compiled as to the uncompiled knowledge. Although this simplifies matters, it probably places limits on the performance gains that can be obtained from compilation. Getting any improvement at all has been a hard job [Minton, 1988],

Because all of the forms B through F have left-hand sides consisting of conditions on states, type A knowledge can be used to conclude those conditions and these inferences can also be compiled:

- $A + X \longrightarrow X$ for X = B, C, D, E, F.

For example, if I have to build a bridge to cross a ravine to get to the office, calculations about its safety go towards a belief that the route using it gets me to work, rather than into the ravine.

5.2 Heterogeneous compilation

Compilation methods resulting in knowledge of types D, E, and F (which I have called condition-action rules, action-utility rules and goals) have received little attention in AI. The following remarks certainly do not constitute compilation algorithms, but serve to indicate some current and future directions for research.

Generating condition-action rules Many of the 'reactive' architectures mentioned above, and production systems in general, use condition-action rules, that identify the conditions under which a given action is expected to be more valuable than all others.[5] These rules can be generated by the following compilation routes:

- $B + F \longrightarrow D$
  For example, if (B) meditating on a full stomach achieves Nirvana, and (F) Nirvana is always desirable, then (D) always meditate after meals. This method is straightforward, and can be simply implemented in an EBL, knowledge compilation [Anderson, 1986] or chunking system [Laird *et al.*, 1986] (it actually constitutes a reasonable characterization of the latter, since every impasse represents a ready-made goal).
- $E + DT \longrightarrow D$ This method is more problematic: conditional knowledge about the absolute and relative utilities of actions must be combined to find the

[5]Production systems with a conflict resolution mechanism do not need to have exactly this strict semantics for their productions, but the same arguments apply.

conditions under which one of them is optimal. In some cases, the utility information for the available actions is provided in parameterized form, allowing the system to compute the ranges of parameter values for which each action is optimal. This approach is common in decision-analytic (particularly multivariate) studies [Fehling and Breese, 1988, Horvitz, 1987, Howard, 1966]. More work is needed to establish efficient and general methods for this kind of reasoning.

- B + C + DT —► D For example, if (B) smoking causes cancer, and (C) cancer is worse than anything, then conclude (D) one should not smoke. In other cases, the utility information will be less absolute, and compilation will be more complex. This compilation method is also problematic because it has to make the kinds of approximations, for the sake of efficiency, that are already hidden in goals and action-utility rules. Consider applying an EBL system to the problem of finding a best action: it needs a proof to the effect that all other actions are guaranteed not to have better outcomes. In non-trivial situations, this proof can be arbitrarily complex — the Intractable Domain Theory problem [Mitchell *et a*l., 1986]. The rule created will have a correspondingly huge number of qualificar tions, and will be essentially useless. A case in point: the concept of a forking move is often cited as the kind of concept that can be learned using EBL techniques, yet the preconditions for guaranteeing that the fork will actually win material, let alone be the best move, are endless.

  The only reasonable solution seems to be to produce condition-action rules that provide an *approximate* guarantee that their recommended action is *more or less* optimal. The rules can then be used as *defaults* in a hierarchy of execution architectures, so that further deliberation of a more explicit nature can over-rule the original recommendation, if time permits. Learning such rules requires sophisticated (or adaptive} accuracy/resource tradeoffs in order to ensure rules that are not too rash yet can execute quickly. Tadepalli [1989] has built a program that learns approximate strategies for king-rook versus king endgames, and exhibits significant speedup as a result. Research is needed to extend such systems to more complex environments. It may turn out that indirect routes, via action-utility rules and goals, are the only feasible approach.

Generating action-utility rules It will often be the case that the value of an action can be estimated, without having a corresponding belief in its optimality. Action-utility rules are compiled from knowledge of the utility of states, and knowledge of the results of actions:

- B + C —>E

This form of compilation seems relatively simple, because it does not need to refer to the utility of *all* available actions, yet there seems to have been little research on automating it. It would be interesting to write a program capable of learning a general rule for estimating the value of a forking move in chess. A chess program constructed using such rules would use them to quickly identify any material-gain or attacking possibilities and to order them for investigation, falling back on full-width search only as a last resort. Similarly, a trading program could learn such rules as "if the current US market price of crude oil is M then buying a cargo of T tons in Venezuela at price P will yield net profit f(M,T,P)" for some known f.

Creating goals The compilation method

- C + DT —► F

essentially finds separable aspects of the utility function that guarantee that achieving a given condition is always a good thing. Even when this guarantee is conditional, the resulting goals may still have enormous computational benefits. For example, the RALPH agents mentioned above can generate a conditional goal to be in the vicinity of food provided no enemies are too near, and can therefore select movement actions with little computational effort. Similarly, a chess player can generate a temporary goal to actively seek a checkmate, thereby allowing a backward-chaining process to find a good strategy-

6 Conclusions

A clear lesson from this brief investigation of compilation is that there is a rich variety of forms of compiled knowledge and of compilation routes. Current methods, with some exceptions as mentioned above, cover only *homogeneous* methods in uniform architectures. Research on building a *mixed* architecture, with a full variety of execution modes using different kinds of knowledge, is a high priority for the RALPH project. One possible application might be in real-time robotics: a robot can learn declarative knowledge about the behaviour of its effectors and the behaviour of objects in its environment, but will need to compile these into effective routines for achieving basic manipulation goals.

When computations are viewed as actions, to be selected according to expected utility just as with ordinary actions in the world, then one can apply decision theory to provide a principled basis for metareasoning [Russell and Wefald, 1988]. One could also apply all of the above analysis to the compilation of metareasoning, especially since it is usually very expensive. An interesting application of such an analysis would be to provide a formal framework for metareasoning in a universal subgoaling architecture [Laird, 1984]. Currently, SOAR converts a fully declarative metalevel model into condition-action rules. Another possibility, that does not seem to be envisaged in SOAR , is to create action-utility rules (type E) for computational actions. This method seems to be effective in controlling computations in a selective search procedure [Russell and Wefald, 1989].

Finally, one can speculate as to why we humans seem to bother with declarative knowledge. Why not learn one of the more compiled forms of knowledge, even learn direct condition-action rules, rather than taking the indirect route? It may be due to a combination of storage requirements and learnability. Essentially, any body of uncompiled knowledge capable of generating a given repertoire of behaviours in a particular environment could be compiled into a network of gated connections between sensors and effectors, with minimal state. The regularities in the network are much more compactly expressible using the declarative knowledge that explains them. But even assuming that space for the complete network is available, learning such a network may be infeasible. Put simply, provided there is *some* regularity in the world, the smaller formulation in terms of uncompiled knowledge will be easier to learn, as shown by standard results in computational learning theory. Moreover, it seems difficult to use prior knowledge in the form of condition-action links to assist in the learning of new condition-action links; how would we construct a radio telescope without any knowledge of the behaviour of the parts or the physics of electromagnetic waves, other than by lengthy trial and error? However, in sufficiently simple task environments (where simplicity depends on the utility function and sensorimotor apparatus, as well as the environment *per se)* the direct approach may succeed. Termites build architecturally sound edifices thirty feet high with no explicit knowledge of anything much. It is an article of faith of declarativists that as the task environment complexity becomes asymptotically high, knowledge will eventually win out over instinct. Finding the cross-over point is currently an empirical task, one we are undertaking in the RALPH project.

References

[Agre and Chapman, 1987] Agre, P. and Chapman, D. (1987) Pengo: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* Seattle, WA: Morgan Kaufmann.

[Anderson, 1986] Anderson, J. R. (1986) Knowledge Compilation: The General Learning Mechanism. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach,* Vol. II. Los Altos, CA: Morgan Kaufmann.

[Brooks, 1986] Brooks, R. A. (1986) A robust, layered control system for a mobile robot. *IEEE Journal of Robotics and Automation,* 2(1), 14-23.

[Buchanan *et ai,* 1979] Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R., Jr. (1979) *Models of learning systems.* Technical report STAN-CS-79-692, Computer Science Department, Stanford University, Stanford, CA.

[Doyle, 1988] Doyle, J. (1988) *Artificial Intelligence and Rational Self-Government* Technical report no. CMU-CS-88-124, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

[Fehling and Breese, 1988] Fehling, M. R., and Breese, J. S. (1988) A computational model for decision-theoretic control of probem-solving under uncertainty. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence,* Minneapolis, MN: AAAI.

[Fikes *et ai,* 1972] Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972) Learning and Executing Generalized Robot Plans. *Artificial Intelligence,* 4, 251-288.

[Horvitz, 1987] Horvitz, E. J. (1987) Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proc. Second Annual NASA Research Forum,* Moffett Field, CA: NASA Ames, 26-43.

[Howard, 1966] Howard, R. A. (1966) Information value theory. *IEEE Transactions on Systems Science and Cybernetics,* SSC-2(1), 22-26.

[Laird, 1984] Laird, J. E. (1984) *Universal Subgoaling.* Doctoral dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

[Laird *et ai,* 1986] Laird, J., Rosenbloom, P., and Newell, A. (1986) Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning 1* (1), 11-46.

[Laird *et ai,* 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987) SOAR: An architecture for general intelligence. *Artificial Intelligence* 33, 1-64.

[Minton, 1988] Minton, S. (1988) Quantitative Results Concerning the Utility of Explanation-Based Learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* Minneapolis, MN: Morgan Kaufmann, 49-54.

[Mitchell *et al,* 1986] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning,* 1, 47-80.

[Newell, 1981] Newell, A. (1981). The knowledge level. *AI Magazine,* 2, 1-20.

[Russell and Wefald, 1988] Russell, S. J., and Wefald, E. H. (1988) *Decision-theoretic control of search: General theory and an application to game-playing.* Technical report UCB/CSD 88/435, Computer Science Division, University of California, Berkeley, CA.

[Russell and Wefald, 1989] Russell, S. J., and Wefald, E. H. (1989) Principles of Metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, Ontario: Morgan Kaufmann.

[Simon, 1982a] Simon, H. A. (1982a) *The Sciences of the Artificial.* Cambridge, MA: MIT Press.

[Simon, 1982b] Simon, H. A. (1982b) *Models of Bounded Rationality, Volume 2: Behavioral Economics and Business Organization.* Cambridge: MIT Press.

[Simon, 1983] Simon, H. A. (1983) Why should machines learn? In Carbonell, J. G., Michalski, R., and Mitchell T., (Eds.) *Machine Learning: an Artificial Intelligence Approach.* Palo Alto, CA: Tioga Press.

[Subramanian and Woodfill, 1989]
Subramanian, D., and Woodfill, J. (1989). Making situation calculus indexical. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, Ontario: Morgan Kaufmann.

[Tadepalli, 1989]
Tadepalli, P. (1989). Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* Detroit, MI: Morgan Kaufmann.