This paper appeared in 1994 in *Cognitive Science*, 18 (3): 479-511.

For the right reasons: The FORR architecture for learning in a skill domain

SUSAN L. EPSTEIN
Department of Computer Science
Hunter College and The Graduate School of The City University of New York

Abstract

The theme of this paper is that knowledge acquisition can be managed as a transition from general expertise to specific expertise. FORR is a general architecture for learning and problem solving that models expertise at a set of related problem classes. This architecture postulates initial broad domain knowledge, and gradually specializes it to simulate expertise in individual problem classes. FORR is based upon a realistic portrayal of the nature of human expertise and its application. Rather than restrict learning to a single method or a single kind of knowledge, the architecture pragmatically requires multiple, disagreeing heuristic agents to collaborate on decisions. A FORR-based program learns both from its apprenticeship to an external expert model and from practice in its domain. An implementation for game playing is described that raises interesting issues about the organization and modification of conflicting expertise, and the role that experience plays in such learning. FORR's principal strengths are its smooth integration of multiple expertise, its ability to learn many ways, its tolerance for human and machine error, its graceful degradation, its transparency, and its support for a developmental paradigm.

INTRODUCTION

There are often many good reasons for doing something. You may, for example, put your foot on the brake of your car while driving because you want to conserve fuel, because there is a pothole in the road, and because you see a police officer ahead. Although no one of those would necessarily cause you to brake, together they make a very good argument for doing so. Just as often, however, those same concerns (cost of operation, cost of maintenance, and protection of privilege) that convinced you to slow down will conflict. This does not invalidate them as operating principles; it just makes decisions more difficult.

Tolerance, even encouragement, of discordant good reasons has several things to recommend it as the foundation of an architecture for problem solving and learning:

• It is human-like. People do not maintain perfectly accurate, consistent knowledge bases.

• It is robust. In an unfamiliar situation there is some general behavior that is reasonable.

• It is efficient. Good reasons focus attention to reduce search.

• It is transparent. Explanations for decisions are readily constructed.

• It is economical. A good reason is usually applicable to more than one problem, even to more than one class of problems.

• It is specialized. A good reason is directly related to the problems it addresses, and is therefore likely to provide greater power than a more general rationale.

FORR (FOr the Right Reasons) is an architecture for general learning and problem solving in skill acquisition based on these ideas. Its “right reasons” are fundamental, potentially conflicting principles for knowledge-based decision in a domain, much like the reasons for braking the car. The contribution of this paper is to justify and demonstrate this perspective for automated knowledge acquisition and to address some classic issues raised by FORR and Hoyle, a program based upon it: granularity, control strategy, learning, and a developmental paradigm.

Granularity in this context is the appropriate specificity for good reasons. Overly general reasons, like economy or good citizenship, may be valid, but their instantiation at decision-making time is likely to be costly, since they are so far removed from the details of the task at hand. Overly specific reasons, like “this particular car comes to a stop from 30 miles per hour in 10 seconds” may proliferate unmanageably because their precision makes them so rarely applicable. One way to view human expertise is as the establishment of a granularity for good reasons that provides high performance while it maintains flexibility. Experts are good at what they do, but in a clearly delineated domain. Within a special set of related problem classes, FORR operates with a granularity established by general knowledge that is theoretically applicable to all of them.

Control strategy in this context is the mediation among good reasons. When the reasons agree, the system can proceed with confidence. When they conflict, however, there must be some way to resolve the disagreement. Part of this control strategy is what people call “commonsense,” compiled ways to adjudicate the disagreement. The rest of the control strategy is more sophisticated, and relies to some extent both on the nature of the domain and the nature of the problem class. For example, commonsense dictates that safety should always supersede preservation of physical property. More specifically, in the domain of driving vehicles, safety of passengers is more important than the integrity of the vehicle. Even more specifically, for the vehicle class of cars, cost of maintenance is more significant than cost of operation, a state of affairs that would not pertain to a class of disposable vehicles. FORR accounts for each kind of control strategy explicitly, and accommodates each one differently.

Learning is essential for any automated expertise. Human expertise is not inbred; it develops with instruction, imitation, and practice. An expert driver probably took a driving course, watched others drive, and practiced. Human expertise is also flexible. Expert drivers can drive many distinct vehicles in the same class without returning to driving school, and they can make wise decisions about those vehicles in many different situations. Human expertise is robust too. Expert drivers can learn to drive unfamiliar vehicles in the same class and even learn to drive vehicles from unfamiliar classes. All of this entails learning. FORR isolates and then integrates instruction, imitation, and practice.

The *developmental paradigm* is the gradual transition from an apprentice program to one that eventually becomes an assistant to its human mentors, a partner, or even a teacher. To support this goal, an architecture must be robust in the face of error and uncertainty, resilient to inconsistency, and able to communicate useful information in a manner people readily understand. FORR provides for each of these concerns.

FORR's thesis is that *there exists broad general knowledge theoretically applicable within certain sets of related problem classes, and that it provides a natural framework from which specific expertise develops.* This approach has several advantages. It offers some immediate, baseline functionality, because it grasps fundamental underlying principles and produces reasonable, if inadequate, behavior. It saves development time, because repetitive common knowledge need be identified and implemented only once for multiple problem classes. It responds robustly to change, because it makes only basic assumptions about the problem classes it encounters.

The next two sections provide a realistic analysis of the nature of human expertise with respect to a class of related problems, and use it to motivate the construction of the FORR architecture. An implementation for game playing is then used to clarify control strategy issues. The final sections argue the cognitive plausibility of this architecture, provide a demonstration of its prowess at an interesting game, and discuss its strengths, limitations, and relation to other work.

THE ORIGIN AND NATURE OF SKILL EXPERTISE

The human expert is associated with a field of endeavor. Such a person is not valued for a broad general intelligence, but for a robust approach to a class of related problems, hallmarked by accuracy, speed, and efficiency. This work focuses on expertise that develops for a *skill domain*, a broad set of related problem classes with the features delineated below. Vehicle driving is used in this section as an example of a skill domain. Driving a car (or a boat or a truck or a bus) is a problem class within that skill. A task, in this example domain, is driving a particular vehicle from one location to another, and a decision point is a moment in time when the driver would have to make a choice about what to do next.

• *Skill domain knowledge is both declarative and procedural.* There is knowledge both about what things are likely to be encountered, like a stop sign, and how to behave in particular situations, like an encounter with a traffic officer (Anderson, 1983; Sussman, 1975).

• *Skill domain behavior holds a set of decisions to a performance standard.* Solutions entail a sequence of choices at a finite number of decision points. It is the collective, cumulative outcome of these decisions that is measured against some performance standard. In vehicle driving, for example, the criterion is timely and safe arrival at some specified destination. The individual decisions during driving are evaluated together, at the end of the performance.

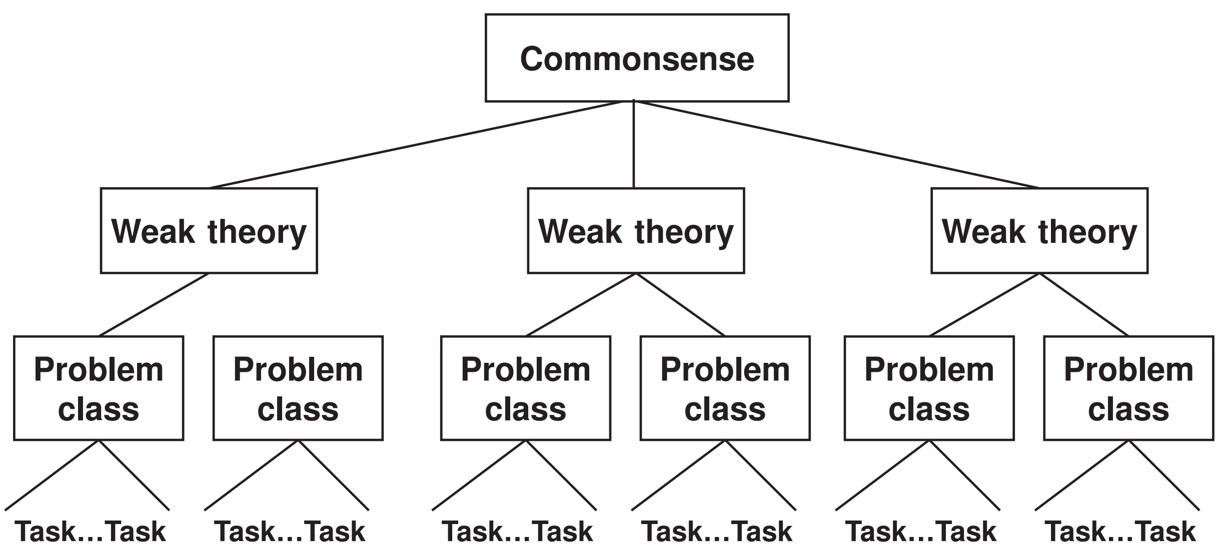
• *Skill domain behavior has ill-defined accountability.* In the sequence of choices during problem solving there is rarely an obvious single action, or even set of actions, that can be credited or blamed for the collective outcome of the behavior. For example, precisely which decision enabled a safe arrival or caused an accident is extremely difficult to pinpoint.

• *Skill domain behavior is learned.* Some of the behavior is learned from instruction, like a drivers' education course. This initial knowledge is then repeatedly modified with experience, like on the road practice. Often the behavior is officially certified when it minimizes deviation from some externally specified performance standard, like the written and road tests for a driver's license.

• *Skill domain behavior is robust.* A skilled individual functions acceptably when confronted with related problem classes. For example, if tomorrow morning in your driveway you were to find a strange vehicle, one you had never encountered before, with a note on it reading "Drive me," you would have some ideas about how to approach that challenge.

Other examples of skill domains include playing a musical instrument, delivering goods, designing a VLSI chip, and playing games. There is no claim here that every problem class can, or should, be couched in this skill framework, only that some domains find this perspective particularly supportive. The definition of a skill domain incorporates Michalski's description of a process learned by repeated practice and correction of deviations from some desired behavior, but does not restrict it to non-symbolic information, subconscious processes, or motor skills (Michalski, 1983).

Figure 1. The intermediate role of the weak theory.



A *weak theory* is broad general knowledge theoretically applicable to each problem class in a skill domain. As Figure 1 suggests, a weak theory is a kind of meta-expert for a skill domain, one that lies between general, problem-solving commonsense and implementations dedicated to a specific problem class. Because it is more specific than commonsense or an expert system shell, a weak theory offers more power; because it is more general than rules directed to a specific task or problem class, it applies to a broader domain. The more commonalities identified in the domain for the weak theory, and the better they are understood and conveyed to a program, the better the initial performance on a new problem class will be, and the more the weak theory can contribute to learning there.

Four kinds of human expert knowledge form the core of a weak theory. (For the philosophical underpinnings, see (Epstein, 1992c).)

• *Problem solving knowledge:* good reasons for making decisions in the domain and a behavioral pattern for how to proceed when working on any task there. The vehicle driving expert, for example, knows about safety, and also knows that the passengers should be secured in the vehicle before it is driven.

• *Start-up knowledge:* information about what problem classes look like in the domain and how to make decisions there. The vehicle driving expert, for example, recognizes a vehicle, and knows under what circumstances certain good reasons should take priority over others.

• *Discovery knowledge:* information on what to learn about individual problem classes. This discovery knowledge can be thought of as the right questions to ask, a set of predetermined features that focus the expert's attention. For example, the expert vehicle driver knows that there is a means of propulsion and a mechanical force to be started in conjunction with it. Such an expert wastes no time considering a strange vehicle's color when learning to drive it.

• *Discovery procedures:* heuristic algorithms to discover problem-class-dependent answers for the right questions already stipulated as the discovery knowledge. The expert vehicle driver, for example, searches for the ignition in the vicinity of the steering mechanism, and, if that fails, in the vicinity of the propulsion mechanism.

There are two kinds of learning relevant to skill domains: learning a skill and learning in a skill domain. *Learning a skill* means acquiring the weak theory itself, the general knowledge applicable to all the problem classes in the skill domain. *Learning in a skill domain* means the gradual instantiation of a weak theory with problem-class-specific data derived from experience. For example, learning to drive a new class of vehicle is learning in a skill domain, while learning about fuel economy and its relative significance to other driving principles is learning a skill. Learning a skill is beyond the scope of this paper. Learning in a skill domain entails the acquisition, validation, and refinement of knowledge for the specified behavior. Such learning provides immediate performance, gradual improvement, and efficient implementation. Learning in a skill domain is the objective of the FORR architecture.

THE FORR ARCHITECTURE

The FORR architecture is predicated on the belief that a program with an adequate weak theory can be applied to any problem class in its domain, where it will initially perform at some basic level and then learn to improve that performance. A FORR-based program works at tasks in a skill domain and learns to make intelligent choices. It develops expertise from a combination of three knowledge sources: its weak theory, its apprenticeship to an expert, and its own problem-solving attempts. This section explains the rationale behind each segment of the design in the context of the preceding discussion on skill domains and expertise. The FORR schematic is assembled here in stages, with the new components at each stage in Figures 3, 4, and 5 darkened for emphasis.

It is important, at this point, to distinguish between absolute expertise and being an expert. *Absolute expertise* is when, from any point in the problem space, one always chooses the best possible alternative. Absolute expertise is derived either from exhaustive search in the problem space or from a complete and correct mathematical theory for the problem space. When absolute expertise is computationally intractable, as it often is in interesting problem classes, the human heuristic alternative is *to be an expert*, i.e., to perform better than most people. “Better than most people” means both faster and with a more highly-valued final outcome (D’Andrade, 1991). Expert programs, however, have traditionally had another constraint imposed upon them: that they make explicit the knowledge that supports their expert performance. A FORR-based program is intended to become an expert, not necessarily to develop absolute expertise.

The basic components

A human expert, as was observed earlier, knows about multiple related problem classes in a domain, how to deal with them, and what decision-making principles underlie them. Figure 2 introduces the first components of the FORR architecture: a problem class definition, a behavioral script, useful knowledge, and the Advisors. Among them, these components describe the domain as a broad but related set of problem classes, provide the combination of declarative and procedural knowledge identified as key to a skill domain, and provide a base level of robustness.

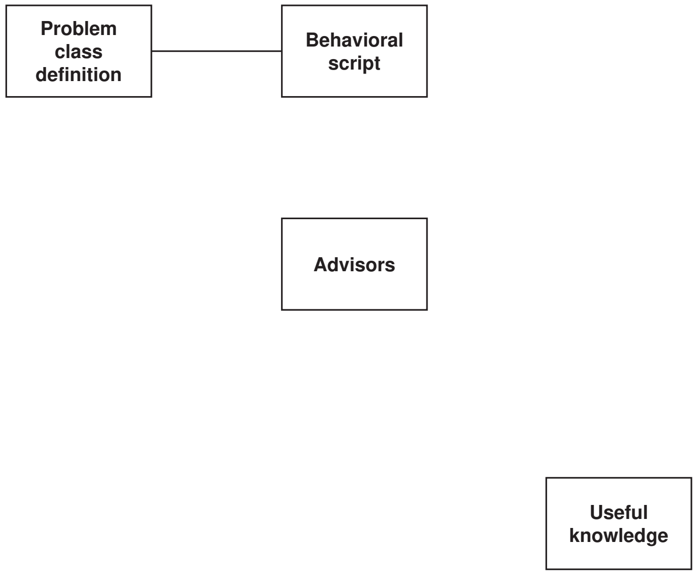
The *problem class definition* is frame-based; it delimits the domain and is part of a human expert’s start-up knowledge. Each problem class is an instantiation of the definition. For the driving domain, for example, cars would be one instantiation of the problem class of vehicles and motorboats would be another.

The *behavioral script* represents the human expert’s problem solving knowledge for how to proceed in the skill domain. It uniformly controls, observes, and describes all of the program’s problem-solving experiences. The behavioral script produces appropriate, but not necessarily intelligent, behavior in the domain. For example, the vehicle driving script begins “Enter vehicle, secure vehicle, secure passengers,…”

Useful knowledge represents the human expert’s discovery knowledge for the skill domain; it is the right questions to ask about a problem class. It is potentially applicable and probably correct. Useful knowledge is stored in a frame with a fixed set of slots that focus the program’s attention.

Initially, the useful knowledge frame for a problem class is empty. For example, useful vehicle driving knowledge includes the location of the ignition and methods of detecting other vehicles.

Figure 2. Representation of a varied domain as a set of identifiable commonalities.



An *Advisor* is a good reason for taking, or not taking, an action. The rationale behind an Advisor may be thought of as a generalization for a set of answers to the question "Why is this a good or bad choice?" There might, for driving, be one Advisor for safety and another for fuel economy. Each Advisor epitomizes a specialized perspective on the decision process, one found generally applicable by an expert in the skill domain.

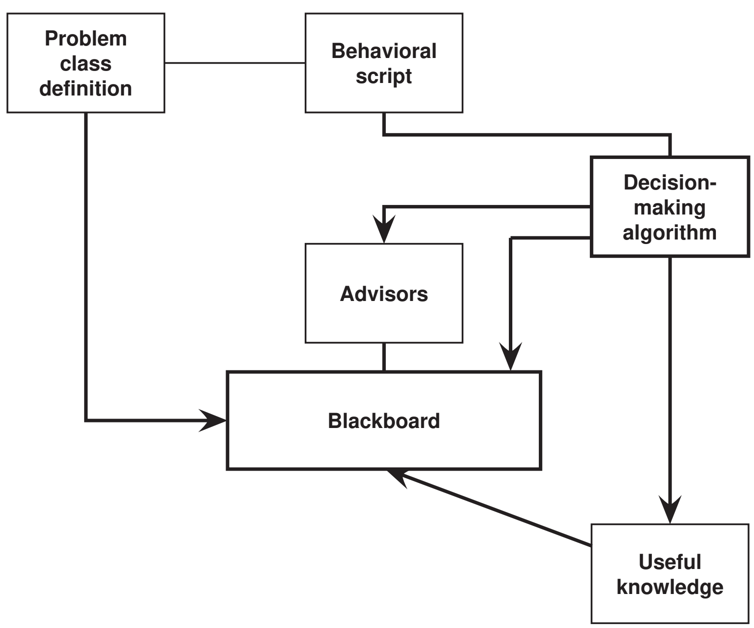
When elicited from an expert, the Advisors, FORR's "right reasons," rarely produce a neat partition of the desired expertise. They may overlap, may conflict, may fail to account for certain portions of the behavior. For example, to base one's driving solely on safety, speed, or fuel economy, would be naive and probably ineffective, but people do manage skilled behavior while integrating such knowledge, and even attribute their expertise to it. Diversity is considered an asset in an Advisor, and conflicting Advisor perspectives are encouraged within FORR. Advisors originate with the system designer and provide significant modularity; it is easy to add new ones as a FORR-based program develops.

All Advisors face exactly the same challenge: given the current state of a task and all the useful knowledge associated with its problem class, provide advice in a uniform format on what action to take next. Each Advisor advocates and opposes possible alternative actions based on its well-defined, if limited, perspective. It produces, at a decision point, any number of comments about what to do next. A *comment* is an ordered triple consisting of the Advisor's name, an action under consideration, and a *strength,* an integer from 0 to 10. A strength of 0 denotes strong opposition to the alternative; a strength of 10 denotes strong support. Although the comments provide a uniform interface between the Advisors and the rest of the FORR architecture, there are no internal uniformity requirements on the Advisors themselves, i.e., the comments may be generated in any manner. An Advisor is merely a resource-limited procedure. When it proposes behavior, an Advisor has read-only access to all the useful knowledge that has been acquired for the current problem class. When learning begins, only the useful knowledge slot names are known. With increased experience, and a commensurate change in the useful knowledge, an Advisor might make different comments about the same state of the world it had faced earlier.

Decision making and learning

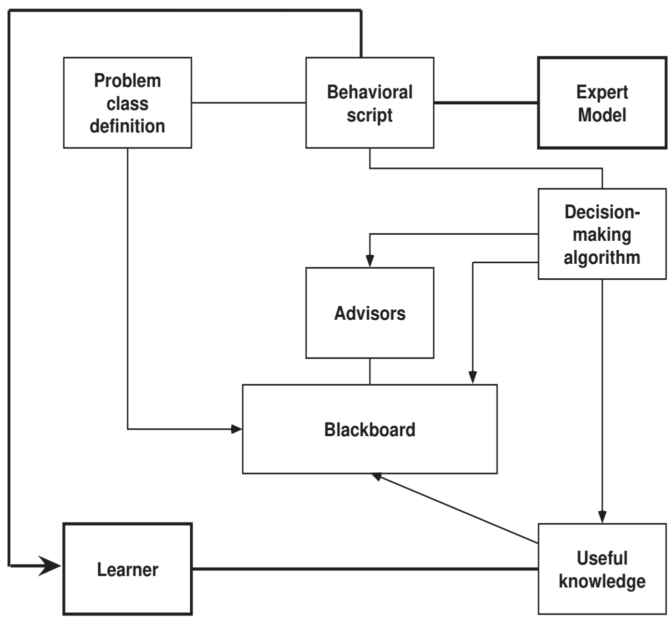
Advisors are also the origin of conflict in FORR, conflict that must be resolved at every decision point. Decision making in FORR, shown in Figure 3, models the features identified for skill domain decisions. The behavioral script activates the problem class definition to post all the current permissible actions in a temporary knowledge repository or *blackboard* (Nii, 1986). The behavioral script then forwards the current problem state to the *decision-making algorithm* which posts the current problem state and the useful knowledge for the current problem class on the blackboard. Next, the decision-making algorithm instructs all the Advisors to read the blackboard and post their comments on it. From the comments on the blackboard, the decision-making algorithm produces a rapid response, as detailed in the next section.

Figure 3. The behavioral script integrates FORR's decision-making process.



The final components of expertise in a skill domain, the Expert Model and the Learner, are introduced in Figure 4. FORR's Expert Model is a human or programmed exemplar of performance, one that is detailed, varied, and external. There is no dialogue between the learning program and its external expert; in particular, the program cannot query the expert. FORR can only watch what the Expert Model does, as reported by the behavioral script. This model may not be perfect, but it is expected to be extremely good at the skill domain, and to provide a broad variety of high quality behavior there.

Figure 4. Learning and external expertise in FORR.



A FORR-based program learns useful knowledge, assembled for each problem class selectively and heuristically from the program's problem-solving experiences. What to learn is determined by the slot names in the useful knowledge frame. How to learn is determined by the *Learner*. At the end of a problem-solving experience the behavioral script forwards a trace to the Learner, which represents a human expert's discovery procedures. The Learner associates with every slot one or more algorithms to compute its appropriate values. The Learner's algorithms have no constraints placed on them by the architecture: they may be inductive or deductive or abductive; they may use explanation-based learning or neural nets or statistical theory.

The full FORR schematic in Figure 4 thus incorporates the three knowledge sources identified for expertise in a skill domain: a weak theory (the problem class definition, the behavioral script, the Advisors, the decision-making algorithm, the Learner, and the useful knowledge), an Expert Model of performance, and experience (attempts by the program to solve skill domain tasks). This also closely correlates with the types of knowledge (domain knowledge, heuristic strategies, control strategies, and learning strategies) Schraagen describes as required for expertise (Schraagen, 1993).

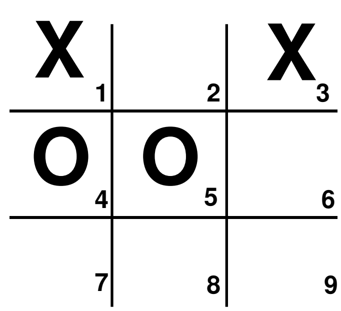
CONTROL STRATEGY FOR CONFLICT RESOLUTION

To explain control in depth, it helpful to consider an implementation. Hoyle is a FORR-based program whose skill domain is two-person, perfect information, finite board games. The boards need not be square, or grid-like, or even two-dimensional. Hoyle differs from traditional game-playing artifacts in several ways (Anantharaman, Campbell, & Hsu, 1988; Berliner & Ebeling, 1989; Samuel, 1963; Tesauro, 1992). It plays many games correctly, i.e., according to their rules. It avoids extensive forward search into the game tree during play, never looking more than two moves ahead. It has no evaluation function or game-specific features. And, unlike most game-playing programs, it learns.

At this writing Hoyle has learned to play each of 18 different games as well as any Expert Model its human mentors can provide. The games in Hoyle's test-bed are culturally diverse and quite varied (Bell, 1969; Zaslavsky, 1982). They were chosen because they span a variety of cultures, and therefore probably capture some aspects of game playing that people find particularly intriguing. Their game graphs lack the complexity of chess or Go, but some of them offer the challenge of cycles and stage transitions, and one of the game graphs has over a billion nodes.

The input to Hoyle is its weak theory for game playing and the definition of the game it is to learn. This weak theory is summarized in the Appendix as a problem-class definition frame, a set of Advisors, a behavioral script, and a useful knowledge frame. The output of the program is an increasingly skilled simulation of playing expertise. Hoyle learns from observation of the external Expert Model with which it competes, and from application of its weak theory to its own behavior and that of the model.

Figure 5. A sample tic-tac-toe state where Hoyle's Advisors could conflict.



Each move decision in Hoyle is based upon conflict among tens, or even hundreds, of comments. Consider Figure 5, a hypothetical example of Hoyle getting advice in a tic-tac-toe contest. It is Hoyle's turn, playing X. As a FORR-based program, Hoyle posts the relevant information on the blackboard: the current game board and whose turn it is to move, whatever useful knowledge for tic-tac-toe is available, and the legal moves (2, 6, 7, 8, and 9) produced when the legal move generator from the problem class definition for tic-tac-toe is called on the state in Figure 5. The good reasons for making a choice could immediately conflict. One Advisor, called Panic, would insist, with a strength of 10, on a move to 6, since O could win there on its next turn. Another Advisor, Worried, would be concerned about future loss in the second column, and recommend moves to 2 and to 8, each with a strength of 9. Worried would also be somewhat less concerned, with a strength of 7, about an eventual loss in the third row, and therefore comment on moves to 7, 8 (again), and 9. Yet another Advisor, Victory, would insist with a strength of 10 that a move to 2 wins immediately

Of course, commonsense dictates that Victory should supersede the others in Figure 5, but the example highlights several important points:

• An Advisor need not be a complex calculator; all it has to do is capture a good reason for behavior.

• Basic conflict resolution in FORR must combine comments to reach a decision.

• Some conflicts can be resolved purely by the rationales behind them.

There are, therefore, two kinds of control in the FORR architecture: prespecified and learned.

FORR's prespecified control partitions the Advisors and confers authority on them within their subset. Advisors are grouped into priority classes called *tiers.* In the first tier perfectly correct Advisors are consulted in a prespecified, fixed order. Some first tier Advisors may have *absolute authority* to make a decision alone. Some first tier Advisors may have *veto power*, the ability to remove a legitimate action from any further consideration, effectively erasing it from the blackboard. When an Advisor exercises absolute authority, not every Advisor in the first tier may have been consulted. Hoyle, for example, has 23 Advisors; Victory has precedence over Panic in the first tier, where they both have absolute authority, and Worried resides in the second tier. Thus the example in Figure 5 was only hypothetical, for once Victory made its comment, the others would never have been consulted. The first tier serves as a filter for decision making; it embodies some commonsense knowledge about how decisions should be made. For example, an Advisor that remembers something should take priority over one that computes the same information. If the first tier makes no decision, one must be constructed heuristically by collaboration in the second.

FORR's learned control is its decision algorithm for the second tier. Unlike the first tier, the second consists of heuristic Advisors without authority; they only make recommendations. Every Advisor in the second tier is consulted before any decision is made. The underlying principle here is that *balancing conflicting heuristic advice is problem-class dependent and should be learned*, i.e., that the deep distinction among problem classes is the way good reasons apply to them during decision making. Some good reasons, of course, are used the same way all the time; those reside in the first tier. Other good reasons, although generally applicable throughout a skill domain, may have properties within a specific problem class. The default decision process, or *fundamental voting paradigm*, for the second tier is simply to tally the comments and take the action with the greatest total strength. Simple variants on this are

- *smoothed voting,* where strengths are converted into “yes” or “no” comments
- *constrained voting,* where only the strongest comment(s) from each Advisor is tallied
- *constrained, smoothed voting,* where only the strongest comment from each Advisor is converted into a “yes” or “no” comment and then tallied

Under all these voting paradigms any ties are broken by random selection.

A FORR-based program can experiment while it considers its burgeoning useful knowledge, trying one voting paradigm or another until it finds a reliable form of expertise, i.e., a good way to resolve conflicts among the right reasons. The four voting paradigms also protect against unintentional bias by the system designer in the way that individual Advisors compute their comments. Comment strengths are intended to reflect only relative value within an individual Advisor’s expertise; they are not intended to reflect any absolute standard with respect to the opinions of other Advisors. If, for example, Advisor A always comments with strengths that are 7’s and 9’s, while Advisor B always comments with 8’s and 10’s, smoothed voting can eliminate that difference. If Advisor C makes many comments, while Advisor D makes few, constrained voting can eliminate that difference. The risk, of course, with smoothed or constrained voting is that knowledge is lost, knowledge that may be necessary for expert performance.

Further refinements to all four of these second tier control strategies are also possible, using relevance and significance. An Advisor is said to be *relevant* if it regularly participates in decisions for its problem class. Although Advisors are intended to be relevant throughout their domain, there may be problem classes in which they make no comment. An Advisor concerned with piece capture, for example, is irrelevant in tic-tac-toe. A FORR-based program can monitor its experience and speed its performance by noting and then consulting only the relevant Advisors for the current problem class. A relevant Advisor is *significant* if it is one that consistently agrees with decisions for the problem class that result in expert outcome. A FORR-based program can identify and record the most consistently significant Advisors in a problem class, and then weight their comments more heavily to improve decisions. Advisor relevance and significance are useful knowledge, continually observed and recorded for use by the decision making algorithm. Note that *a FORR-based system credits and blames the reasons behind decisions (the Advisors), not the individual decisions themselves.* This is intended to produce a broader expertise, not for a particular task, but for the problem class as a whole. Although it has Advisors instead of sensors, FORR’s control approximates what Clancey has termed a “dialectic coupling of sensorimotor systems in an ongoing sequence of coordination” (Clancey, 1993).

DISCUSSION

FORR's principal strengths are its smooth integration of multiple expertise, its ability to learn many ways, its tolerance for human and machine error, its graceful degradation, its transparency, and its support for the developmental paradigm. The discussion below is drawn from several years of laboratory experiments with Hoyle, during which it has learned to play 18 games extremely well. (See, for example, (Epstein, 1993) and (Epstein, 1994).) These experiments have been directed to a variety of issues in machine learning. At least ten tests were run with each game; many of the games have been learned hundreds of times under varying conditions.

• **Multiple expertise**: The Advisors constitute multiple, if imperfect, experts. Their organization into what Brooks calls a subsumption hierarchy accounts for FORR's modularity (Brooks, 1991). One can specify a new Advisor with very minimal consideration of its impact on the control structure. A new Advisor need only be established in an appropriate tier. For example, in its first *capture game* (where the number of playing pieces one participant has on the board in the next state may be reduced by the other participant's move selection), Hoyle saw no reason to prefer a move that captured pieces. A new Advisor with that perspective now encourages Hoyle to minimize the material held by the other participant and maximize its own. In non-capture games, this Advisor makes no comments, and Hoyle learns to ignore it.

• **Multiple learning methods**: FORR can acquire useful knowledge with any kind of learning algorithm; each item of useful knowledge is associated with one or more learning procedures. For example, Hoyle learns forks by explanation-based learning, openings by rote, expert responses by induction, and significant states by deduction. (For further details, see (Epstein, 1990; Epstein, 1992b).) Unrestricted to a single learning method, a FORR-based program can even have multiple learning algorithms for the same slot. The learning algorithms produce data that, regardless of its method of acquisition, is uniformly available to the Advisors as a basis for their comments. Thus the output of the various learning methods is integrated, rather than the processes themselves.

• **Error tolerance**: Errors are not an issue in FORR. No comment is considered an error, even though it may be wrong from the omniscient perspective of the entire search space. As useful knowledge is acquired, the collective opinions of the Advisors should eventually outweigh a misguided comment. If a heuristic should learn incorrect useful knowledge, either because the Expert Model or the learning heuristic errs, the Advisors, again with the backing of more recent useful knowledge, can eventually override it through their comments. For example, Hoyle has learned to imitate an expert's incorrect move in a particular situation, but after much subsequent play deduced that it was an error, and then refused to repeat it.

• **Graceful degradation**: Through its specification of general domain knowledge, FORR guarantees a minimal performance standard. Hoyle, for example, can play any game in its domain according to the rules, if not expertly, given the game definition and its game-playing behavioral script. Even at a problem class it does not yet solve well, or with a newly-encountered Expert Model, the decisions of a FORR-based program reflect the sensible underlying premises of its weak theory.

• **Transparency**: A FORR-based program not only learns to make expert decisions, it can explain why it makes them in a given situation and what it knows overall about a given problem class. This clear distinction between control knowledge and declarative knowledge in FORR supports quick debugging and user-friendly visibility. Individual decisions are made because of the comments on the blackboard and the voting paradigm used; they may be displayed during execution or recomputed if the useful knowledge has not changed in the interim. One aspect of knowledge about a problem class is the relevance and significance of the individual Advisors. Hoyle, for example, can retrieve the three most significant Advisors for a particular game and then produce advice for a human player who accumulates useful knowledge. Consider this translation of tic-tac-toe's significant Advisors: "To play tic-tac-toe well, defend carefully against the simple plans of the other participant (Worried), forward your own simple plans (Candide), and forward as many offensive plans simultaneously as possible (Greedy)."

• **Developmental paradigm**: A FORR-based program can be used as a collaborative development environment, a cycle where the Expert Model represents the best human knowledge about a problem class until the program finds the flaws. For example, the games Hoyle plays come from anthropology books that specify the rules, but rarely give any indication of correct strategy or an inherent advantage or disadvantage to the one who moves first. In our laboratory we play a new game intensively and then build an initial Expert Model program that represents our best intuition about the game. This is usually a very fallible "expert;" Hoyle often learns to play better than this first pass. Once we see how and why Hoyle wins, we improve the Expert Model, and the cycle repeats. Some games have had several increasingly strong Expert Models before we are satisfied that they represent absolute expertise. Thus Hoyle has become a research partner in the discovery of knowledge about a new game.

FORR's principal limitations are its reliance on human intelligence, reliance on experience, lack of creativity, potential memory requirements, and validation of its own expertise. The construction of a FORR-based program clearly relegates important tasks to the human system designer: the correct description of an appropriate behavioral script, the identification of the perspectives that determine the Advisors, the assignment of Advisors to tiers based upon knowledge about relations among their perspectives, the specification of which Advisors access which useful knowledge, and the description of how they apply that knowledge. When carefully cast, these tasks are applicable to all problem classes in the domain. The one-time effort to construct them is a good investment, however, and FORR's modularity ensures that any modifications necessitated by human oversight are quick and easy.

Reliance on experience driven by an external expert can be limiting. Recent work has demonstrated the substantial impact that the nature of the Expert Model can have on what is learned, how quickly it is learned, and how useful that learned knowledge is (Epstein, 1992a). As it acquires useful knowledge, for example, Hoyle's play may become somewhat routine and unadventurous. It may appear to have learned to play expertly, but an opponent who makes different, perhaps less skilled, moves could still readily defeat it. At this point the program needs to practice, to broaden its experience base, to think about plausible situations as well as the most challenging. The solution for Hoyle is to play against other versions of itself. This is a supplementary experience to what might be an imperfect or overly rigid Expert Model.

The rigidity of the useful knowledge slots and the prespecification of their learning algorithms, however, is more problematic. The long-range performance of a FORR-based program is directly dependent on the quality of the useful knowledge it captures, because that knowledge serves as input to the Advisors and affects the quality of their advice. Without a uniform representation for useful knowledge, like those in SOAR or PRODIGY, there is no obvious way to generate new useful knowledge items (Carbonell, Knoblock, & Minton, 1991; Rosenbloom, Newell, & Laird, 1991). It is possible, however, to specify multiple ways to learn the same useful knowledge, label the results differently, and create distinct Advisors that propound the various learning theories.

Other than the Learner's discovery algorithms, there is currently no mechanism in FORR for knowledge revision or forgetting. There could eventually be too much useful knowledge if the learning algorithms were not sufficiently selective. This has not yet been a problem for Hoyle, but it is a potential one for any large search space.

There are two sources of non-determinism in FORR: random tie breaking during voting and stochastically engendered variety in an automated Expert Model. A FORR-based program therefore acquires knowledge that varies in quantity and content from one run to the next. In experiments with Hoyle, neither learning time, nor memory allocation for useful knowledge, nor consistent expert performance, nor stability of the knowledge base has proved a fool-proof indicator of competence. A FORR-based program is intended to learn, but can offer no clear and reliable signal of when it has achieved adequate performance, for it has no absolute goal to reach.

FORR is an architecture for limited rationality. There is no expectation that a FORR-based program ever begins as an expert in any problem class, or that it eventually achieves perfect or optimal performance. Learning algorithms retain only part of what is experienced. Second tier Advisors have resource limits associated with them; all comments must be produced within those limits. Instead of perfection, a FORR-based program offers real-time reasonable behavior, gradual improvement in problem solving, and the ability to adapt to a changing environment.

COGNITIVE PLAUSIBILITY

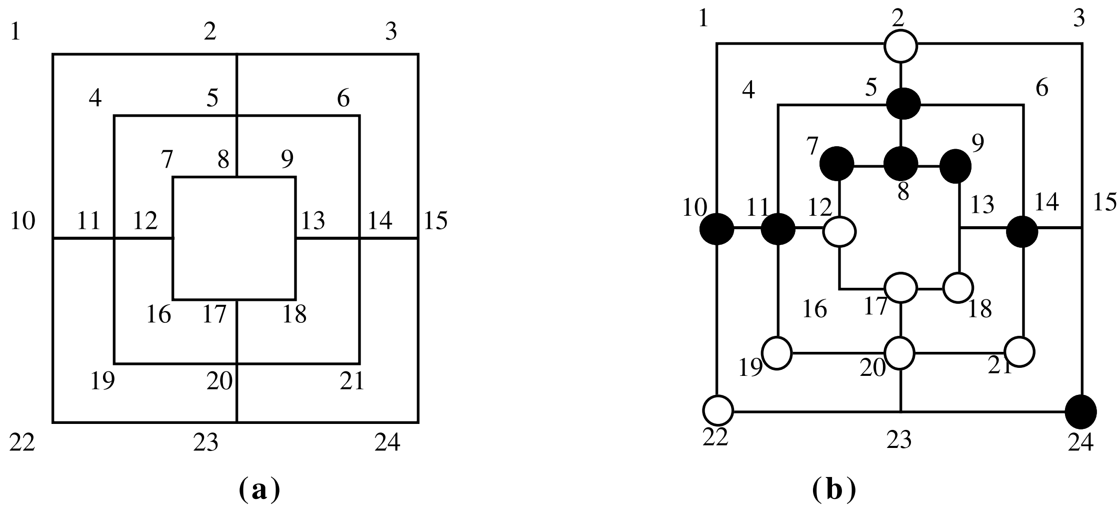
Empirical studies of expertise support the cognitive plausibility of the FORR architecture. Absolute expertise is not an issue: “an expert is someone capable of doing the right thing at the right time” (Holyoak, 1991). There is also evidence against a single, correct human way to be an expert (Holyoak, 1991). Instead, an instantiated weak theory is precisely those “processes and knowledge specific to the particular domain” that human experts have been observed to have (Ericsson & Smith, 1991). The deliberate omission of extensive search during decision making is consistent with the general problem-solving literature on human behavior (Charness, 1991). Both induction and the instantiation of the useful knowledge frame are also supported by the literature (Holyoak, 1991). Data from recent studies support the hypothesis that experts, unlike novices, process multiple solution constraints in parallel rather than test and reject hypotheses serially (Novick & Coté, 1992). Recent evidence for a parallel, rather than a serial, stream of consciousness, supports this approach too (Dennett & Kinsbourne, 1992). Although Hoyle’s second tier is implemented on a serial machine, it effectively parallelizes this segment of the decision process.

Empirical studies of expertise also support the cognitive plausibility of the choice of games as a test domain and portions of Hoyle’s weak method. Many authors have cited the need to work in a domain with accepted, well-defined evaluation metrics, like game playing (Charness, 1991; Ericsson, et al., 1991). Hoyle’s significant states, and all the Advisors that reference them, associate a solution method as part of the immediate comprehension of the task, as human experts do (Ericsson, et al., 1991). The Hoyle Advisor on openings simulates observed human behavior too (Charness, 1991). What is clearly lacking from Hoyle is knowledge representations relevant to perceptual chunks, for both storage and analysis. That is now being addressed in current research.

A DEMONSTRATION

The strongest argument for FORR as a working model is Hoyle’s recent simulation of some intelligent behavior at a game it has just begun to learn. The decisions Hoyle makes must be attributed to the collective wisdom of its individual Advisors, who together behave quite cleverly. Nine men’s morris is played on the board in Figure 6(a). Each participant has nine playing pieces of one color (black or white), and black plays first. Pieces may rest only on the intersection of two or more lines; there are 24 such *positions*. Nine men’s morris has two consecutive stages, a placing stage and a sliding stage. In the *placing stage,* the initial board is empty, and a turn consists of placing one’s unplayed piece on a position. Once all 18 pieces are on the board, the *sliding stage* begins, and a turn consists of sliding one’s piece along a line to the next empty position. A playing piece may not jump over another or be lifted from the board during a slide. Three pieces of the same color in a straight line along any side of any square (e.g., 1-2-3 or 7-12-16) is called a *mill.* Each time a participant achieves a mill, she immediately removes a piece of the opposite color that is not in a mill. If all pieces of the opposite color are in mills, any such piece may be removed. Removed pieces are permanently captured; they never return to the board. The first one reduced to two pieces or unable to move loses.

Figure 6. Nine men's morris

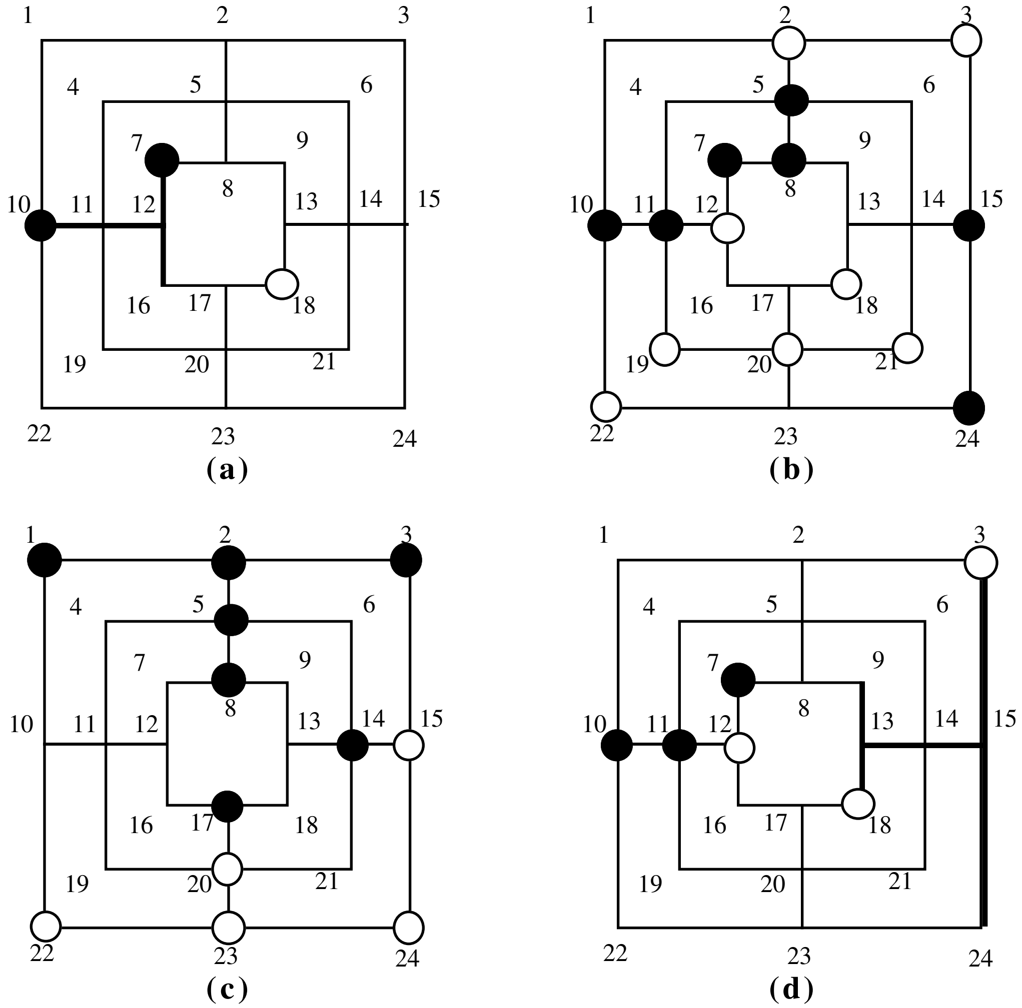


Nine men's morris has approximately 143 billion nodes in its game tree, the largest Hoyle has attempted thus far. There are an average of 15.5 legal moves for every turn in the placing stage, and an average of 7.5 in the sliding stage. To appreciate the power of the examples that follow, the reader should examine carefully the simple Advisor descriptions in Table 4 of the Appendix; they are the sole premises upon which Hoyle bases its move decisions. Remember, too that at no time during play does any Advisor ever look more than two moves ahead. Figures 6(b) and 7 display board positions from contests that Hoyle played after learning in only ten contests against a hand-crafted expert program. Although this expert program is imperfect, it plays extremely well.

Hoyle knows how to block a mill. Situations like Figure 6(b) arose even during early learning. Hoyle was playing black in the sliding stage (some pieces had already been captured), and it spotted trouble in the making. On its next turn, white was very likely to slide from 20 to 23; then it would be able to move its piece back and forth from 17 to 20, making a mill at 19-20-21 and capturing a piece on every second turn. Once the juggernaut began, black would lose pieces repeatedly until it lost the contest. Hoyle moved from 24 to 23, preventing that very long and unpleasant sequence. A human observer might claim that Hoyle anticipated the repeated opening and closing of the 19-20-21 mill six times so that Hoyle would be reduced to two pieces. Inspection of the comments at this point, however, reveals that Hoyle chose this move because the

Advisors Greedy and Mobility swung an otherwise close vote, not because Material was farsighted.

Figure 7. Good decisions in nine men's morris



Hoyle learns, very quickly, how to fork. The construction of two or more potential mills at once is called *forking,* and Hoyle learns many clever ways to do so. Figure 7(a) is an example of a simple fork on two mills in the placing stage. Hoyle opens at 7, white replies at 18, and Hoyle decides to move to 10. If white leaves 11, 12, and 16 open, Hoyle can play 12 on its next turn, thereby simultaneously threatening two potential mills (7-12-16 and 10-11-12), only one of which can be prevented. With 10 Hoyle gives the appearance of a five-move plan with boolean expressions and a don't-care value (meaning any move would be acceptable): "10, not 11 or 12 or 16, 12, don't-care, 11 or 16." There is no five-ply lookahead or represented plan, and Hoyle had never seen this state before; all it knew was a little about openings. Among them the Advisors identified an appropriate move. In another sequence similar to this one, where a human player as white failed to notice the threat and did not play in either potential mill, Hoyle took the equivalent of 12 immediately, demonstrating that it not only knew how to threaten forks, it knew how to execute them.

Hoyle behaves as if it had game-dependent subgoals. The goal in this task is to play well, i.e., to win whenever possible. Since a win can occur when the opposition is reduced to two pieces, capturing pieces is a subgoal. Making a mill is the way to capture pieces, and therefore is also a subgoal. Although Hoyle cannot reason in this fashion and has no representation for these plans or any subgoals, it often plays as if it does. Simple two-ply look-ahead can make Hoyle wary, so that it behaves as if it were subgoaling on mills. In Figure 7(b) Hoyle, as black, is a piece behind (due to the 19-20-21 white mill), still in the placing stage. It has successfully executed the fork on 5, 7, and 8, however, against a human opponent. White has just defended at 2 and Hoyle is about to reply with 9, but the program must also decide which white piece to capture. It removes 3 because the Advisors recognize the potential both for a 1-2-3 white mill and a 3-15-24 mill of Hoyle's own. Material sees a state with one less piece for white, and the potential for another black mill, as a definite improvement. (This contest will be continued shortly.)

A still stronger play in the placing stage is detailed in Figure 7(c). There is a five-move plan, restricted to the center square, which can be executed if and only if one side occupies 8 and 17 when 7, 9, 12, 13, 16, and 18 are empty. (The contents of the other positions are irrelevant.) The plan is known as a depth 3 fork, because it simultaneously threatens more than one fork, not just more than one mill (Epstein, 1990). The plan may be paraphrased as follows:

- play a corner, (7, 9, 16, or 18)
- white is forced to block (9, 7, 18, or 16, respectively)
- play the corner opposite white's last move (16, 18, 7, or 9, respectively)
- white is confronted with a fork, and has two choices, neither of which will prevent a mill

(Because each side has lost pieces during placing, this plan actually has gaps in it to allow for sliding pieces into position.) One possible instantiation of this plan is the move sequence 9, 7, 18, 16, 13, but, with the fork at the end, there are a total of eight different five-move plans inherent here. Figure 7(c) is a crucial moment; it is Hoyle's turn as white and, although there are 12 legal moves, only six of them, played now, will prevent the onslaught. Hoyle plays 9, demolishing the fork. This could have been the luck of the odds in the face of a voting tie; however, inspection of the comments assures us that the Advisors actually preferred this move, even though Hoyle's fork Advisor did not participate. Once again the program's decision is right, based upon the right reasons, but without the human explanation, planning, or extensive search.

We now return to the contest that began with a fork in Figure 7(b), and continue an opening sequence which had not been played, even symmetrically, in Hoyle's prior experience. The expert is forced to defend against the fork at 12, and Hoyle responds with a not particularly strong move to 11. In Figure 7(d) the expert has begun, with a play at 3, a depth three fork that simultaneously proceeds toward mills at 13-14-15, 3-15-24, and 9-13-18. This fork is directly related to white's piece at 18 and requires positions 9, 13, 14, 15, and 24 to be empty. Hoyle plays 24, which not only destroys white's plan but also forces the expert to defend at 22 (else Hoyle will fork on mills at 1-10-22 and 22-23-24). From the perspective of a five-ply search, there were two excellent reasons for Hoyle's play at 24, but there *was* no five-ply search; Advisors called Greedy and Mobility actually forced this move selection. The right reasons, shallow though they appear, sufficed with a two-ply search.

RELATED WORK

Despite some superficial similarities, there are substantial differences between FORR and a traditional production system. At first glance the Advisors might appear to be productions and the decision-making algorithm some prespecified control strategy. In a production system, however, each rule may trigger (indicate its appropriateness) in exactly one way, and exactly one rule is selected to fire (execute). In FORR, an Advisor may, and often does, hold itself to be appropriate in many ways, and represent each of those ways as a comment. An Advisor may be thought of as a meta-rule that is instantiated both by the task it addresses and by the useful knowledge available to it. Unlike production rules, Advisors have resource bounds and signal when they exceed them. In addition, the actual decision may match any number of comments, as if it were firing any number of rules. A FORR-based program seeks not the right rule, or even the right rule-generator, but the right decision based on consensus. Unlike most production systems, part of FORR's control strategy is both problem-class dependent and learned. Finally, FORR does not just respond to the current state of the world; it relies on accumulated useful knowledge to comment and to decide.

FORR's most radical feature is its reactivity. For the premise that "true intelligent behavior arises out of a unified reasoning system" (Carbonell, et al., 1991), it substitutes the idea that the responses from many small individual agents can be coordinated reflexively to simulate intelligent decision making (Brooks, 1991). FORR differs from the classical reactive system, however, in its insistence on explicit concept representation (Epstein, 1992c).

Most general problem solving and learning architectures like PRODIGY, SOAR, and Theo are designed to learn how to specialize themselves to solve problems. SOAR does forward chaining in multiple problem spaces and then incrementally compiles search control decisions from traces of problem solving as production rules called "chunks" (Laird, Rosenbloom, & Newell, 1987). PRODIGY uses means ends analysis in a search space and then learns explanations that select, reject, or prefer rules (Carbonell, et al., 1991). Theo tries an ordered list of inference methods in sequence to learn slot values (Mitchell, et al., 1991). The primary similarities among these architectures and FORR is their commitment to a uniform representation for knowledge (rules in SOAR, PDL logic in PRODIGY, frames in Theo and FORR) and an indexing of the problem solving knowledge to the problem description. The primary differences between these three and FORR are their subscription to the unified architecture hypothesis that FORR rejects, and their response to uncertainty with extensive search instead of FORR's reasonable but heuristic guesses. Here are some other distinctions:

• *Representation of control knowledge:* SOAR represents learned control knowledge as operator preferences, Theo as a sorted list of methods to calculate slot values, and PRODIGY as selection, rejection, and preference rules. FORR represents control knowledge by the way Advisors are relegated to and empowered in tiers.

• *Discovery knowledge:* Soar and Theo always learn; PRODIGY directs its learning with the utility principle. FORR's weak theory determines what to learn and when to learn it.

• *Multiplicity of learning methods:* SOAR has a single method. Theo and PRODIGY each have several, with elaborate systems used to integrate and rank them. FORR simply associates one or more methods with each slot. SOAR learns chunks, PRODIGY learns explanations, but a FORR-based program could learn a chunk for one slot, an explanation for a second, and weights for a neural net for a third.

• *Caching of experience:* SOAR and Theo retain everything they calculate from experience, PRODIGY retains preferences with a high utility factor, and FORR retains only what its selective Learner's algorithms specify.

• *Explicit explanations:* PRODIGY learns explanations, Theo computes them and stores them with computed slot values, and SOAR offers none. FORR's explanations for its decisions are the discordant comments of its Advisors.

• *Beliefs about problem classes:* Unlike the others, FORR does not require the information collected by the Learner to be correct, only applicable in future decisions. Both Theo and FORR deliberately construct knowledge applicable to entire classes of problems (not problem states); the others do not.

• *Transparency:* PRODIGY and Theo make all knowledge available to all components; SOAR's chunks are unavailable for examination. FORR takes a middle ground: it shares useful knowledge among its Advisors but restricts problem-solving experiences to the learner and restricted comment computation to the authoring Advisor.

• *Response to dynamic environment:* SOAR and PRODIGY expect that the domain in which they learn is static. FORR, and to some extent Theo, acknowledge that "truth" may change and that learned knowledge may have to be revised.

The idea of a weak theory, although not the term itself, has several precedents. BASEBALL used general knowledge, such as how to focus on traces and the nature of competition and cooperation, to extract and abstract general concepts, like hit and out, from the training instances (Soloway, 1978). The program, applied only to baseball, embodied a declaratively represented weak theory about certain kinds of games in its analysis, while Hoyle's is both declarative and procedural. AM, with its fixed frame slots and procedures for filling them, had a weak theory for mathematical discovery, but it was not required to produce solutions or perform to meet an external standard, only to generate declarative knowledge (Lenat, 1976). EURISKO, which learned to play at least one difficult game superbly, was intended to modify its own weak theory, but its problem classes had few similarities on which to capitalize (Lenat, 1983).

There has been much work in general methods for learning about how to play games. Minton used constraint-based generalization to deduce winning combinations from a single example in tic-tac-toe, go-moku, and chess (Minton, 1984). His program learned segments of Hoyle's Advisor Pitchfork. Tadepalli's program naively constructs king-pawn endgame move sequences in chess that are overly general, and then refines them when they fail (Tadepalli, 1989). These plans are more general than Hoyle's highly restricted ones. Unlike Hoyle, which attempts to learn from every contest, Tadepalli's program learns only when one of its plans fails, and cannot use symmetry. Collins and others have explored adaptive decision making in tic-tac-toe and chess to learn Advisors, part of the weak theory (Collins, Birnbaum, & Krulwich, 1989). Flann has described a knowledge-compilation method to learn problem-instance/best-action pairs that cover much of a game subtree with abstractions (Flann, 1992). The resultant incomplete theory is different from a weak theory in that it does not purport to cover the entire search space of a problem class, as a weak theory does, and so will find itself at a loss about 20% of the time (Flann, 1990).

Overall, Hoyle certainly begins with more prior general knowledge than these four game-learning systems. Hoyle's greater prior knowledge, however, is in keeping with its weak theory architecture and accounts for the speed with which it learns and the expertise which it ultimately develops without any game-specific features or evaluation functions. Hoyle's Advisors are, admittedly, more opaque than the typical explanation-based reasons, but the contents of the game library are not. For example, the significant Advisors for a game can be used to produce explanations that humans find quite accurate.

Four other sophisticated game-learning programs have been tested on a few of the simple games that Hoyle learns easily. Henri, a line-oriented pattern-recognition program, and N-N/Tree, a neural network program augmented with search and temporal difference learning, do not learn to play perfect tic-tac-toe, even after 1000 training contests (Flax, Gelfand, Lane, & Handelman, 1992; Painter, 1993). Dooze, a classifier system, learns tic-tac-toe in 63 contests using 150 classifier rules, but does not scale up to games with many positions because it must begin with a large number of rules to describe the legal moves (Esfahany, 1992). A program that acquires strategic threat-and-defense patterns (a variant on Morph) learns to play after 250 contests and stores 50 patterns, but must be hand-coded with game-specific concepts for tic-tac-toe, since it was designed for chess. By comparison, Hoyle learns to play tic-tac-toe perfectly in, on average, 16 contests without any game-specific concepts. Each of the other programs learns to specialize itself by tailoring its pattern generalizations to a single set of game-specific (Morph, N-N/Tree) or board-specific (Dooze, Henri) algorithms. As a result, its memory requirements grow dramatically with the number of positions on the board. Hoyle only stores on average 3 significant states, 5 openings of one move each, and 4 previously-played contests to guide its play. Such an approach is unlikely to scale up for a large board game like Go.

Hoyle's global approach to a domain with a set of independent knowledge sources is reminiscent of a distributed system. Each knowledge source is implemented as an agent (an Advisor) with a common goal but is heuristic, and its comments may be inaccurate. Hoyle's agents do not negotiate; they act together because the centralized blackboard control strategy forces their collaboration when it combines their advice into a single choice recommendation (Levesque, Cohen, & Nunes, 1990). This coordination relies upon a high-level strategic plan for advice sharing, similar in spirit to that of Corkill and Lesser (Corkill & Lesser, 1983). The plan, however, is partially predetermined and partially learned, and agents take only minimal directions, in the form of restricted options on the blackboard, from each other. Each Hoyle agent spends most of its time in computation rather than communication, as do those in DARES (Conry, MacIntosh, & Meyer, 1990). Unlike DARES, however, lack of direct communication frees Hoyle's individual agents to use powerful, even idiosyncratic, knowledge representations that support efficient reasoning from a particular viewpoint. The hierarchical nature of Hoyle's decision making process stems not from Durfee and Montgomery's levels of abstraction and preassigned authority values, but from clusters of authority values, its tiers (Durfee & Montgomery, 1990). The tier in which an agent lies is predetermined by the nature of the skill domain, but the program learns a kind of relative authority within the second tier so that, unlike most distributed problem solvers, performance improves with experience.

CONCLUSIONS

The task of a FORR-based program is to apprentice, to acquire useful knowledge about each problem class in a skill domain, possibly relevant and probably correct knowledge that is meaningful to its human mentors and enables it to perform as an expert would. Hoyle demonstrates that a FORR-based program is possible, and can be efficient, powerful, and robust. What is noteworthy about Hoyle is not the (medium level) difficulty of the games it plays, but the fact that, from its own experience, it *learns* to play them, first learns effectively, then learns efficiently, repeatedly learns beyond its mentors' skill.

In FORR's version of the developmental paradigm, *learning is the gradual instantiation of a weak theory with problem-class-specific data derived from experience.* This instantiation is derived from prespecified knowledge about what to learn and how to learn it. There is no requirement that the weak theory itself be correct or complete, nor that it perform well on problems immediately. If the computer apprentice never acquires absolute expertise, it can still participate in the developmental paradigm: serve as a human backup, offer advice, and contribute its useful knowledge. If the computer apprentice eventually outperforms its human guides, it can still continue to improve and evolve, with or without guidance. Because FORR treats its acquired knowledge only as useful, rather than certain, it tempers inductive optimism with experience, and countenances inconsistencies as differences of opinion. Hoyle learns different control strategies for different problem classes, a demonstration that a weak theory can support, without in advance embodying, knowledge sufficient for expertise in the domain.

A FORR-based program's development under resource-limited, experience-driven analysis is a different, and promising, way to examine a search space. Although it begins at each game as a novice, Hoyle learns quickly, offers increasingly strong competition, and often acquires and displays clever methods for winning previously unknown to its opposition. A FORR-based program soon moves from defaults to coherent knowledge, based upon evidence drawn from its experience.

FORR shows how *independent knowledge sources with a common goal can capture a powerful global view.* Diversity in Advisors, in problem classes, and in tasks, is an asset. As new problem classes are encountered, any needed new Advisors, new useful knowledge slots, or new associated learning procedures are easy to add. A weak theory is intended to evolve into a set of experts, not begin as one. This evolution is envisioned as a realistic partnership among skilled humans, the system designer, the end user, and the program. The modularity of a FORR-based program supports the gradual development of automated expertise.

FORR strikes a balance between costly, perhaps intractable, exhaustive deduction and the less trustworthy leaps of faith demanded by induction from relatively few examples. Because the architecture supports a set of related problem classes simultaneously, it makes development and learning cost-effective. A FORR-based program begins with predefined areas of uncertainty and defaults, gathers evidence to refine its knowledge base, seeks to validate what knowledge it acquires, and learns problem-class-specific conflict resolution. FORR's modularity makes it decomposable, easy to analyze, understand, and maintain. Hoyle demonstrates how a FORR-based program can take controlled problem-solving risks to learn a small but powerful amount of useful knowledge that supports the simulation of expert performance.

FORR does not rely on a single representation or learning strategy. Each item of useful knowledge can be represented in a way that makes it most efficient and flexible. Each learning algorithm can use the method that best satisfies its requirements. Experience in each problem class can develop support the development of an appropriate control strategy.

There is no claim that FORR would be the best architecture for every task. The ideal application is in a skill domain, one with a set of related tasks susceptible to the same general problem-solving knowledge. Each task should select from among a finite number of choices at a finite number of decision points (although voting on a continuous spectrum of choices would be a simple extension). A set of generally valid expert principles should be identifiable for favoring or discouraging individual choices at decision points. There should be access to a human or programmed Expert Model, and there should be a standard to evaluate the overall outcome of decision making in the domain. Finally, since FORR is predicated on gradual development of expertise, the domain should be one that readily tolerates error, one where failure lies within reasonable resource bounds. For development in the complex and ill-understood skill domain of game playing, FORR works well. Exploration of several other domains is now underway; robot path-finding on college campuses, managing a baseball team, and Japanese-English language translation are all FORR-based programs under development.

FORR is founded on the idea that there are commonalities that support expertise in a skill domain. Although this weak theory does not provide immediate expertise in a new problem class, it supports the learning of that expertise. The right reasons for taking actions across a skill domain are usually fairly clear, and the conflict among them can be used productively. Encouraging the Advisors to make multiple comments, and then balancing them against each other by voting, avoids a precise commitment as to which perspective should take priority at which moment in any given task. The tier hierarchy provides a reasonable foundation and then conflict resolution takes over. The complexity comes in balancing the right reasons against each other, and balancing those reasons for control appears to be learnable. All it takes is a weak theory, an expert to watch, and some practice.

ACKNOWLEDGMENTS

This work was supported in part by NSF 9001936 and PSC-CUNY 668287. Portions of this material were refined at the Machine Learning 92 Workshop on Computational Architectures for Supporting Machine Learning and Knowledge Acquisition. The author thanks Tom Mitchell and Jack Gelfand for many insightful discussions on this work.

APPENDIX: HOYLE'S WEAK THEORY

A FORR-based program's weak theory includes its problem-class definition frame, a behavioral script, a set of Advisors, a useful knowledge frame, and a set of learning algorithms for the acquisition of useful knowledge. This Appendix describes Hoyle's weak theory for two-person, perfect information, finite board games. Additional information and the learning algorithms are available in (Epstein, 1992b).

Hoyle's Problem-Class Definition

Hoyle defines a game as any instantiation of its problem class definition. In Table 1 the definition has been instantiated for tic-tac-toe. The definition is the slot names; the instantiation is the slot values and their association with those slots. Five variables and nine functions completely define the game. Only the non-empty slots are shown for tic-tac-toe; more complex games may also have values stored for a list of predrawn lines on the game board, an adjacency graph of positions, and functions to map back and forth between a list representation of the game board and a two-dimensional plot of the positions and their contents (Epstein, Gelfand, Lesniak, & Abadie, 1993).

TABLE 1
An Instantiation of the Problem Class Definition

Name: *tic-tac-toe*
Token for Player: *X*
Token for Opponent: *O*
Dimension: 3×3
Initial-board: *(NIL NIL NIL NIL NIL NIL NIL NIL NIL)*

Directions for user: *directions-tic-tac-toe*
Move input reader: *reader-tic-tac-toe*
Move filter: *legalp-tic-tac-toe*

Display function for current state: *display-tic-tac-toe*
Move effector: *effector-tic-tac-toe*
Legal move generator: *generator-tic-tac-toe*

Predicate to detect end of contest: *endp-tic-tac-toe*
Predicate to calculate winner: *winp-tic-tac-toe*
Predicate to calculate loser: *lossp-tic-tac-toe*

Hoyle's Behavioral Script

Hoyle's behavioral script enables it to play any two-person, perfect information, finite board game, when it applies slot values from the game definition. A pseudocode version of this algorithm appears in Table 2.

TABLE 2
Hoyle's Behavioral Script For Game Playing

```
Select a game
Initialize the game frame
For some number of contests                 *tournament length or leave unspecified*
  Determine the identities of Player and Opponent
  Initialize the board
  Do while the contest is not over
    If it is the keyboard's turn              *non-Hoyle mover*
      then   do until a legal move is input
               request a move
      else   generate all legal moves         *Hoyle to move*
             case:
               there are no legal moves: resign
               there is exactly one legal move: make it
               there is more than one legal move: consult the Advisors to select one
    Record the most recent move in the contest's history
  Announce the result of the contest          *winner or draw?*
  Conduct a postmortem on the contest
```

Hoyle's Useful Knowledge

Hoyle's useful knowledge frame has slots that focus attention during discovery. Each slot is associated with one or more procedures in the Learner. The value of a slot is computed when the Learner uses data provided from Hoyle's experience. Table 3 shows a hypothetical instantiation of the useful knowledge frame for tic-tac-toe. A *significant state* is a game state from which there is a certain winner, even though the contest is not yet over; a fork is a meta-plan for multiple ways to win (Epstein, 1990). Only the non-empty slots are shown for tic-tac-toe; more complex games may also have values stored for *dangerous states* (not proved significant within the time limit but seen in the context of a loss), an associative pattern database, and some distance measurements on the two-dimensional version of the game board (Epstein, et al., 1993).

TABLE 3
Hoyle's Useful Knowledge For Game Playing

Expected role winner: *draw*
First mover has ever won: *true*
Second mover has ever won: *false*
Applicable two-dimensional symmetries: *all*
Reliable voting paradigms: *all*

Openings: *1 is a win, 3 is a draw, 5 is a draw, 7 is a draw, 9 is a draw*
Forks: *none*
Significant states: *state-1, state-2, …, state-10*
Histories: *contest-1, contest-2, contest-3, contest-4*
Average contest length: *9*
Expert moves: *none*

Relevant Advisors: *all*
Most significant Advisors: *Candide, Worried, Greedy*

Hoyle's Advisors

Hoyle's Advisors provide advice on move selection when it is Hoyle's turn, and their implementation is game-independent. There are currently 23 Advisors, grouped into two tiers. They are described in Table 4, with the first tier Advisors in their specified, implemented order. If the first tier produces a single move recommendation, then the second tier is not consulted. Because Hoyle plays a broad variety of games, certain Advisors may not be relevant to a specific game.

TABLE 4
Hoyle's Advisors For Game Playing

Name	**Tier**	**Description**	**Authority**	**Applies Useful knowledge**
Wiser	1	Makes the correct move if the current state is remembered as a certain win.	Absolute	Yes
Sadder	1	Resigns if the current state is remembered as a certain loss.	Absolute	Yes
Victory	1	Makes the winning move from the current state if there is one.	Absolute	No
Don't Lose	1	Eliminates any move that will result in an immediate loss.	Veto power	Yes
Panic	1	Blocks a winning move the non-mover would have if it were his turn now.	Absolute	Yes
Shortsight	1	Advises for and against moves based on a two-ply lookahead.	Absolute	Yes
Enough Rope	1	Avoids blocking a losing move the non-mover would have if it were his turn now.	Veto power	No
Anthropomorph	2	Moves as a winning or drawing non-Hoyle expert did.	None	Yes
Candide	2	Formulates and advances naive offensive plans.	None	No
Challenge	2	Moves to maximize its number of winning lines and minimize the other's.	None	No
Coverage	2	Maximizes the mover's markers on predrawn lines and minimizes the non-mover's.	None	No
Cyber	2	Moves as a winning or drawing Hoyle did.	None	Yes
Greedy	2	Moves to advance more than one winning line.	None	No
Leery	2	Avoids moves to a state from which a loss occurred, although limited search proved no certain failure from it.	None	No
Material	2	Moves to increase the number of its pieces, or decrease those of the other.	None	No
Freedom	2	Moves to maximize the number of its subsequent immediate moves and minimize those of the other.	None	No
Not Again	2	Avoids moving as a losing Hoyle did.	None	Yes
Open	2	Recommends previously-observed expert openings.	None	Yes
Patsy	2	Recreates visual patterns credited for positive outcomes in play and avoids those blamed for negative ones.	None	Yes
Pitchfork	2	Advances offensive forks and destroys defensive ones.	None	Yes
Shortcut	2	Bisects the shortest paths between pairs of markers of the same contestant on predrawn lines.	None	No
Vulnerable	2	Reduces the non-mover's capture moves on two-ply lookahead.	None	No
Worried	2	Observes and destroys naive offensive plans of the other participant.	None	No

REFERENCES

Anantharaman, T., Campbell, M. & Hsu, F. (1988). Singular Extensions: Adding Selectivity to Brute Force Searching. In *Proceedings of the AAAI Symposium on Computer Game Playing. AAAI 1988 Spring Symposium Series*. 8-13.

Anderson, J. R. (1983). Acquisition of Proof Skills in Geometry. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Ed.), *Machine Learning: An Artificial Intelligence Approach* (pp. 191-219). Palo Alto: Tioga Publishing.

Bell, R. C. (1969). *Board and Table Games from Many Civilizations* . London: Oxford University Press.

Berliner, H. & Ebeling, C. (1989). Pattern Knowledge and Search: The SUPREM Architecture. *Artificial Intelligence*, *38*(2), 161-198.

Brooks, R. A. (1991). Intelligence without Representation. *Artificial Intelligence*, *47*(1-3), 139-160.

Carbonell, J. G., Knoblock, C. A. & Minton, S. N. (1991). PRODIGY: An Integrated Architecture for Planning and Learning. In K. VanLehn (Ed.), *Architectures for Intelligence* (pp. 241-278). Hillsdale, NJ: Lawrence Erlbaum.

Charness, N. (1991). Expertise in Chess: The Balance between Knowledge and Search. In K. A. Ericsson, & J. Smith (Ed.), *Toward a General Theory of Expertise - Prospects and Limits* (pp. 39-63). Cambridge: Cambridge University Press.

Clancey, W. J. (1993). Situated Action: A Neuropsychological Interpretation (Response to Vera and Simon). *Cognitive Science*, *17*(1), 87-116.

Collins, G., Birnbaum, L. & Krulwich, B. (1989). An Adaptive Model of Decision-Making in Planning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 511-516.

Conry, S. E., MacIntosh, D. J. & Meyer, R. A. (1990). DARES: A Distributed Automated REasoning System. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, 78-85.

Corkill, D. D. & Lesser, V. R. (1983). The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Kaufmann, 748-756.

D'Andrade, R. G. (1991). Culturally Based Reasoning. In A. Gellatly, & D. Rogers (Ed.), *Cognition and Social Worlds* Oxford: Clarendon Press.

Dennett, D. C. & Kinsbourne, M. (1992). Time and the Observer: The Where and When of Consciousness in the Brain. *Behavioral and Brain Sciences*, *15*, 183-247.

Durfee, E. H. & Montgomery, T. A. (1990). A Hierarchical Protocol for Coordinating Multiagent Behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, 86-93.

Epstein, S. L. (1990). Learning Plans for Competitive Domains. In *Proceedings of the Seventh International Conference on Machine Learning*. Ed. B. W. Porter, & R. J. Mooney. Morgan Kaufmann, 190-197.

Epstein, S. L. (1992a). Learning Expertise from the Opposition - The Role of the Trainer in a Competitive Environment. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*. Morgan Kaufman, 236-243.

Epstein, S. L. (1992b). Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems*, *7*, 547-586.

Epstein, S. L. (1992c). The Role of Memory and Concepts in Learning. *Minds and Machines*, *2*, 239-265.

Epstein, S. L. (1993). *The Hoyle Training Experiments* (CS-TR-93-01). Department of Computer Science, Hunter College of The City University of New York.

Epstein, S. L. (1994). Toward an Ideal Trainer. *Machine Learning*, *15*(3).

Epstein, S. L., Gelfand, J., Lesniak, J. & Abadie, P. (1993). The Integration of Visual Cues into a Multiple-Advisor Game-Learning Program. In *Proceedings of the AAAI Fall Symposium on Games*. 92-100.

Ericsson, K. A. & Smith, J. (1991). Prospects and Limits of the Empirical Study of Expertise: An Introduction. In K. A. Ericsson, & J. Smith (Ed.), *Toward a General Theory of Expertise - Prospects and Limits* (pp. 1-38). Cambridge: Cambridge University Press.

Esfahany, K. A Pattern Classifier that Learns to Play Games. In preparation.

Flann, N. (1990). Applying Abstraction and Simplification to Learn in Intractable Domains. In *Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 277-285.

Flann, N. S. (1992). *Correct Abstraction in Counter-Planning: A Knowledge Compilation Approach*. Ph.D., Oregon State University,

Flax, M. G., Gelfand, J. J., Lane, S. H. & Handelman, D. A. (1992). Integrating Neural Network and Tree Search Approaches to Produce an Auto-Supervised System that Learns to Play Games. In *Proceedings of the 1992 International Joint Conference on Neural Networks*.

Holyoak, K. J. (1991). Symbolic Connectionism: Toward Third-Generation Theories. In K. A. Ericsson, & J. Smith (Ed.), *Toward a General Theory of Expertise - Prospects and Limits* (pp. 301-336). Cambridge: Cambridge University Press.

Laird, J. E., Rosenbloom, P. S. & Newell, A. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, *33*(1), 1-64.

Lenat, D. B. (1976). *AM: An Artificial Intelligence Approach to Discovery in Mathematics*. Ph.D., Department of Computer Science, Stanford University,

Lenat, D. B. (1983). EURISKO: A Program That Learns New Heuristics and Domain Concepts. *Artificial Intelligence*, *26*(1-2), 61-98.

Levesque, H. J., Cohen, P. R. & Nunes, J. H. T. (1990). On Acting Together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, 94-99.

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Ed.), *Machine Learning: An Artificial Intelligence Approach* (pp. 83-134). Palo Alto: Tioga Publishing.

Minton, S. (1984). Constraint-Based Generalization - Learning Game-Playing Plans from Single Examples. In *Proceedings of the Fourth National Conference on Artificial Intelligence*. William Kaufmann, 251-254.

Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M. N. & Schlimmer, J. C. (1991). Theo: A Framework for Self-Improving Systems. In K. VanLehn (Ed.), *Architectures for Intelligence* (pp. 323-356). Hillsdale, NJ: Erlbaum.

Nii, H. P. (1986). Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine*, *7*(2), 38-53.

Novick, L. R. & Coté, N. (1992). The Nature of Expertise in Anagram Solution. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum, 450-455.

Painter, J. (1993). *Pattern Recognition for Decision Making in a Competitive Environment*. Master's, Hunter College of the City University of New York,

Rosenbloom, P. S., Newell, A. & Laird, J. E. (1991). Toward the Knowledge Level in Soar: The Role of the Architecture in the Use of Knowledge. In K. VanLehn (Ed.), *Architectures for Intelligence* (pp. 75-112). Hillsdale, NJ: Erlbaum.

Samuel, A. L. (1963). Some Studies in Machine Learning Using the Game of Checkers. In E. A. Feigenbaum, & J. Feldman (Ed.), *Computers and Thought* (pp. 71-105). New York: McGraw-Hill.

Schraagen, J. M. (1993). How Experts Solve a Novel Problem in Experimental Design. *Cognitive Science*, *17*(2), 285-309.

Soloway, E. (1978). *Learning = Interpretation + Generalization: A Case Study in Knowledge-Directed Learning*. Ph.D., University of Massachusetts,

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition* . New York: American Elsevier.

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*. Morgan Kaufmann, 694-700.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, *8*(3/4), 257-277.

Zaslavsky, C. (1982). *Tic Tac Toe and Other Three-in-a-Row Games, from Ancient Egypt to the Modern Computer* . New York: Crowell.