Capitalizing on Conflict: The FORR Architecture

Susan L. Epstein
Department of Computer Science
Hunter College and The Graduate School of The City University of New York
695 Park Avenue, New York, NY 10021

Abstract

FORR is a general architecture for learning and problem solving. Its design harnesses multiple, disagreeing perspectives on a domain and learns to make intelligent choices from their conflicting advice. FORR has been implemented in Hoyle, a program that learns to play two-person, perfect information board games. A FORR-based program is modular, robust in the face of error or lack of information, degrades gracefully, and should improve gradually with problem solving experience. This paper describes a theory for skill acquisition, the knowledge base FORR requires, and the way FORR's design affects learning.

1. INTRODUCTION

Human experts incorporate many diverse considerations and constraints in their behavior. Rarely do these individual components produce a neat partition of the desired expertise. Instead they may overlap, may conflict, may fail to account for certain portions of the behavior; some, in their compiled form, may elude the notice of even the most astute observer. For example, a vehicle driver has fundamental considerations, like safety, speed, and fuel economy, that are always operative and influence most driving decisions. To rely on any one of these rationales would be naive and probably ineffective, but people do solve problems while integrating such knowledge, and even attribute their success to them.

When AI programs simulate human expertise, they typically represent expert constraints and considerations as rules or procedures *(behavior proposers)* each of which recommends a specific action. These behavior proposers are accompanied by prespecified strategies designed to prevent conflict among them. FORR (FOr the Right Reasons) is an architecture for learning and problem solving in a broad domain of related problem classes. Rather than forestall conflict among behavior proposers, FORR actively encourages conflict and capitalizes on it. FORR's behavior proposers, called Advisors, typify the right reasons for making decisions in a domain. A FORR-based program makes each decision by resolving conflicts among the opinions of its Advisors. Such a program learns task-specific useful knowledge that both supports conflict and refines its resolution.

FORR has been applied to game playing in Hoyle, a program that learns to play two-person, perfect information board games (Epstein, 1992b). To date, Hoyle has become an expert at 12 such games. The ability to acquire expertise for an entire class of problems requires a theory of skill acquisition, presented in the next section. The remainder of this paper focuses upon how FORR shapes what Hoyle learns.

In reference to Hoyle, it is important to distinguish among a game, a contest, and a tournament. A *game* is an activity with a board, playing pieces, and rules. A *contest* is an experience playing a game, beginning at an initial situation specified by the rules and ending when the rules declare a winner or a draw. A *tournament* is a sequence of contests at a specific game in which the participants alternately go first or second. Thus chess is a game at which two people might play a tournament of 16 contests.

2. A THEORY OF SKILL ACQUISITION

Absolute expertise is when one performs perfectly, i.e., from any state in the problem space one makes the best possible decision. Absolute expertise can be derived either from exhaustive search in the problem space or from a complete and correct mathematical theory for the problem space, such as those developed by for some games (Berlekamp et al., 1982) . When exhaustive search is prohibitive and there is no complete and correct theory for the domain, absolute expertise is computationally intractable. The human heuristic alternative is *relative expertise*, performance better than that of most people, here called simply "expert." A human's expertise is usually judged only on relative ability to perform, without any standards for the description and sharing of expert knowledge. This may certify as expert one who can perform in the problem class but cannot explain that behavior to others, a state of affairs with which expert systems developers are all too well acquainted. For a program to be accepted as an expert, however, it must not only perform well but also make explicit the knowledge that supports its performance.

Expertise in one problem class is often linked to expertise

in another. A *skill* is a behavior which satisfies the following criteria:

- *Varied domain:* Its domain encompasses a broad but related set of problem classes.
- *Dual representation:* It relies upon both fundamental declarative knowledge about the domain and a set of general procedures (Anderson, 1986; Sussman, 1975) that generate actions.
- *Collective evaluation:* The collective, cumulative outcome of these actions is measured against some performance standard for expertise.
- *Learning requirement:* It is initially taught and then repeatedly modified with experience to minimize deviation from the performance standard.
- *Robustness:* It functions acceptably when confronted with related problems.
- *Ill-defined accountability:* In a trace of the behavior, there may be no obvious single action, or even set of actions, to credit or blame for the collective outcome.

Examples of a skill include driving vehicles, playing keyboard instruments, delivering goods, designing VLSI chips, and playing games. This definition 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 (1983).

In this context it is important to distinguish between two kinds of learning. *Learning a skill* means acquiring the general knowledge applicable to all the problem classes in the skill domain. *Learning in a skill domain* means acquiring the knowledge specific to a problem class in the skill domain. Both of these entail the collection, validation, and refinement of knowledge for the specified behavior.

People begin with rudimentary abilities that support their learning, such as vision, balance, and synopses of experience labeled "commonsense." A program to learn in a skill domain should have such fundamental general knowledge too. Just as people apprentice to an expert, a program to learn in a skill domain can be provided with the kind of instruction (initial data and procedures) that an apprentice receives (e.g., Mitchell, 1985; Steinberg, 1987). Finally, just as people acquire expertise partly from observation of expert performance and partly from their own attempts, a program to learn in a skill domain can observe an expert and attempt to solve problems itself. Thus a machine should learn a skill from a combination of:

- *Expert information:* identifiable general problem-solving knowledge for a broad skill domain and a set of possibly conflicting expert principles
- *Observation of an expert:* a human or programmed model of performance, one that is expert, detailed, varied, and external
- *Attempted performance:* repeated attempts whose overall behavioral outcome is compared with that of the expert model

FORR is an architecture for learning in a skill domain from those three sources.

Just as an expert system shell postulates fundamental principles for knowledge-based decision, a *weak theory* contains general knowledge theoretically relevant to behavior in all the problem classes in its skill domain. Thus a weak theory for a skill domain is a kind of meta-expert. Weak theories lie between general problem-solving and implementations dedicated to a specific problem class. (See Figure 1.) 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 problem or problem class, it applies to a broader domain. Although a weak theory derives its structure and fundamental principles from general problem-solving knowledge, it offers an intermediate range of specialization that partially bridges the gap to the task-specific implementation. Thus a weak theory may be seen as a collection of commonalities for a skill. The more commonalities identified in the domain, and the better they are understood and conveyed to a program, the more the weak theory can contribute to learning there.

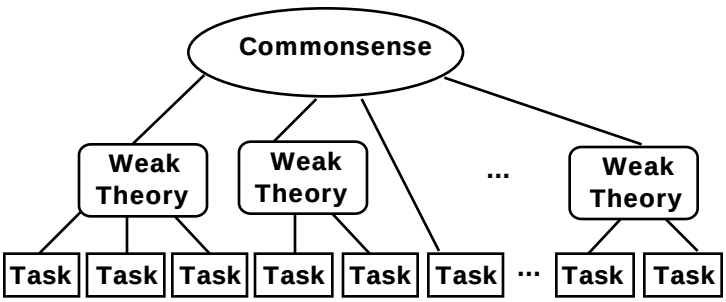

Figure 1. The intermediate role of the weak theory.

By definition, a weak theory for a skill domain must support both problem-solving and learning. Initially a program can work on any problem in its domain simply by applying its weak theory. At this point it will probably solve problems inadequately, but it will behave within the domain's bounds of acceptability. For example, a program with a weak theory for driving vehicles might make erroneous suggestions but it will not attempt to disassemble a vehicle if that is forbidden by the weak theory. For a given problem class in the skill domain, *learning is the gradual instantiation of the weak theory with problem-specific data derived from experience.*

A weak theory must specify:

- *Start-up knowledge:* what is already known about the domain
- *Problem solving knowledge:* how to proceed when solving any problem in the domain
- *Discovery knowledge:* what to learn about individual problem classes
- *Discovery procedures:* how to learn about individual problem classes

The start-up knowledge in a weak theory is either applicable to all problem classes or limited to problem definitions. For example, it might define an object class called "vehicle class" and might also instantiate it with data. Start-up knowledge does not, however, provide statistics on how reliably the vehicle class performs under certain conditions; that is to be learned with experience.

Insistence on general knowledge about the domain requires a breadth of focus more familiar to those who construct expert system shells than those who build expert systems. Our experience has been that the effort required to specify general knowledge pays off handsomely as the problem classes encountered within the domain become more difficult and less well known to the user. There is no requirement that the weak theory itself be correct or complete, nor that it perform well on problems immediately. 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. A program based upon a weak theory both modifies itself by learning and supports its modification by others. In our experience, such a program's failures are the impetus for human refinement of the theory. If, in some problem space, the program eventually outperforms its human guides, it can still continue to improve and evolve, with or without guidance.

3. FORR, THE ARCHITECTURE

The FORR architecture embodies the theory of skill acquisition detailed in the previous section. A FORR-based program is intended to learn to simulate a skill. Its goal is open-ended learning; there is no internal representation to judge when learning should halt. In theory, there is always more to learn.

A schematic of the FORR architecture appears in Figure 2. FORR uses the three sources predicated for skill learning in the previous section: expert information, expert observation, and attempted performance. FORR's expert information is a domain-dependent but task-independent initial knowledge base consisting of a problem class definition, a behavioral script, a set of Advisors, a useful knowledge frame, and a Learner. FORR's behavioral script observes an external expert model and drives its attempts at expert performance.

FORR also specifies all the segments of a weak theory. Start-up knowledge in FORR is the problem definition and an expert model, human or programmed, of the behavior to be simulated. Problem solving knowledge in FORR consists of a behavioral script and a set of behavior proposers intended to minimize or eliminate search. Discovery knowledge in FORR is called useful knowledge; it is a frame-based representation of what to learn about individual problem classes. Discovery procedures stipulate how to learn about individual problem classes; there is one for each useful knowledge slot.

FORR assumes the intrinsic correctness of its weak theory for a domain. Every problem class must be definable within the definition frame, and must have some model of expert behavior, although the model need not be flawless. The actions dictated by the behavioral script must be appropriate to every problem class. The Advisors and the discovery procedures must be theoretically applicable to every problem class.

FORR's frame-based *problem class definition* has fixed slots that determine the kind of tasks the system can address. For example, Hoyle's problem class definition identifies the rules and material used in a game. A task in FORR is an instantiation of the problem class definition; for example, one of Hoyle's tasks is tic-tac-toe. The *behavioral script* uniformly controls, observes, and describes all of the program's problem-solving experiences. It produces appropriate, but not necessarily intelligent, behavior in the domain. For example, Hoyle's behavioral script controls, moderates, and referees all contests. The behavioral script also provides observation-only access to an external *expert model* at the task to be learned.

Each of the *Advisors* epitomizes a specialized perspective on the decision process, one found generally applicable by an expert in the skill domain. 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/bad choice?" For example, an Advisor called Victory might

recommend a move to Hoyle because it achieves a win. All of Victory's advice has the same reason behind it, regardless of the game or the current game state. An Advisor need

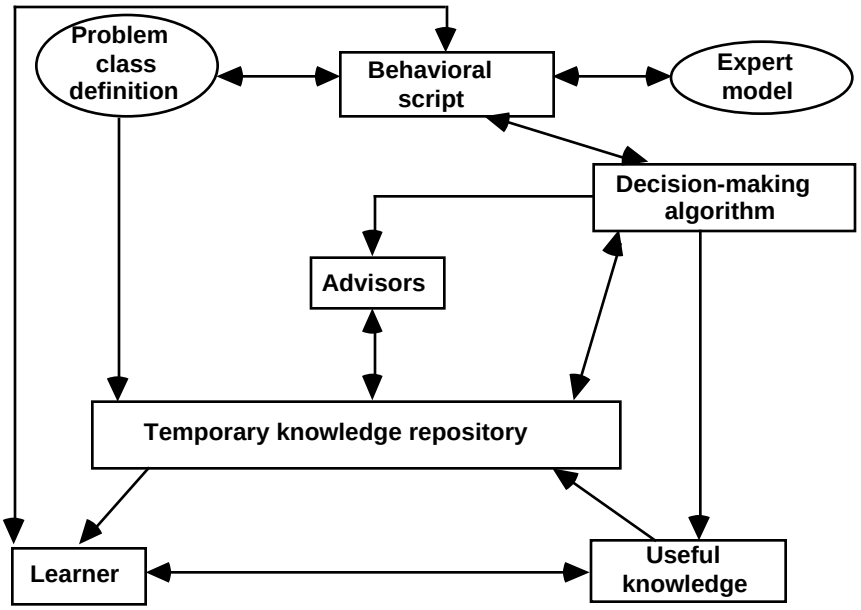

Figure 2. The FORR architecture.

not always be correct; it is only expected to capture a good reason for taking, or not taking, actions. 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. Resource bounds may be part of an Advisor's heuristic nature. No uniformity of representation or process among Advisors is required. Diversity is considered an asset, and conflicting Advisor perspectives are encouraged by the architecture.

Useful knowledge is potentially applicable and probably correct. It is stored in a frame with a fixed set of slots that focus the program's attention. Useful knowledge is assembled for each task selectively, heuristically, from traces of problem-solving experience. The Advisors access whatever useful knowledge has been acquired when they propose behavior. Before learning, only the useful knowledge slot names are known. Hoyle, for example, has an initially empty slot for whether or not it is an advantage to go first in a game. The *Learner* is a set of heuristic algorithms to discover entries for the useful knowledge slots. For example, Hoyle's Learner runs tournaments of contests at a game and processes the resulting data to extract slot values.

An Advisor remarks on possible behavior in a *comment*, a permissible action with a *strength* that describes a spectrum from adamant opposition (low value) to insistent support (high value). At any decision point, each Advisor may make any number of comments about any number of actions. For example, when it is Hoyle's turn to move, the comments recommend or advise against legal moves from the current game state. The *decision-making algorithm* decides what to do based on the frequency and weight of the comments for and against each behavior. There is a default decision-making algorithm and a set of useful knowledge slots whose values can modify it.

Conflict in FORR arises from the differing opinions presented in the Advisors' comments, and is resolved by tabulating the information they contain. As part of the start-up and problem-solving knowledge, the Advisors are partitioned into priority classes called *tiers*. When the program must decide among a set of actions, it consults the tiers in turn. If any tier produces an absolute recommendation or reduces the possible legal behaviors to a single one, none of the subsequent tiers is consulted. The first tiers are reserved for Advisors that are absolutely correct in their opinions. Control in the first tiers is hard-coded to reflect domain-wide commonsense knowledge there, and typically consults Advisors in a fixed order. Not every Advisor may be consulted on every decision. Some Advisors may have absolute authority (full decision making power) or veto power (ability to reject an action). If the filter of the first tiers fails to make a decision, one is heuristically constructed from the collective opinions of the Advisors in the last tier. Although the last tier has a default control procedure, the relative importance of the Advisors may depend upon the problem class, i.e., some perspectives may be more significant than others in

particular problem classes. Such control knowledge can be acquired through experimentation and experience.

What FORR learns is slot values for useful knowledge. As observed above, some of this is control knowledge for the decision-making algorithm; the rest is analysis of prior experience that influences the formation of the comments from which decisions are made. How FORR learns a slot value depends upon the learning algorithm associated with the slot. Any blend of valid machine learning approaches is acceptable. The accuracy of the information extracted from experience is determined by the implemented learning algorithms. The subsequent appropriate application of the learned knowledge is the responsibility of the Advisors.

4. HOYLE, THE PROGRAM

Hoyle is a FORR-based program with 15 Advisors. Thus far Hoyle has learned to play 12 games as well or better than its human mentors, by playing against an external (usually programmed) expert and analyzing traces of those contests. The games in HOYLE's testbed are culturally diverse and quite varied (Zaslavsky, 1982; Bell, 1969). 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.

Each move decision in Hoyle may be based on the conflict among tens, or even hundreds, of comments. Hoyle's default conflict resolution is a decision-making algorithm that simply tallies the comment strengths and selects the move with the greatest associated strength. Ties are broken by random selection. Hoyle also tests alternative conflict resolution algorithms in play, ones that smooth comment strengths and/or limit each Advisor to a single strongest comment. Hoyle also tabulates how often each Advisor comments for a particular game, and how often those comments appear to support expert behavior. From all this useful knowledge, Hoyle can tailor a conflict resolution algorithm for each game, one that is more resource-efficient as well as more adept at move selection.

Hoyle learns from *contest traces*, move-by-move descriptions of play from start to finish. These traces serve as input to the Learner's algorithms. They focus search on fragments of the game tree, provide examples of expert behavior (the external model's moves), and provide data for analysis. Examples of Hoyle's slot values are whether or not it is an advantage to move first in a contest, applicable two-dimensional symmetries, and average contest length.

5. LEVERAGE AND LIMITATIONS

FORR's principal strengths are its modularity, its ability to learn many ways, its tolerance for human and machine error, its graceful degradation, and its transparency. The organization of the Advisors into a subsumption hierarchy accounts for FORR's modularity. One can specify a new Advisor, i.e., an entire class of rules, 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 markers 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 its opponent's markers. A new Advisor with that perspective now encourages Hoyle to minimize the material held by the other participant. In non-capture games, this Advisor makes no comments, and Hoyle learns to ignore it.

Like THEO, FORR can acquire useful knowledge with any kind of learning algorithm (Mitchell et al., 1990). For example, Hoyle can learn useful knowledge by EBL, by induction, by deduction, and by neural net. Unrestricted to a single learning method, FORR can specify the most appropriate or most efficient algorithm for each slot.

Errors are not an issue in FORR. No behavior proposal is considered an error, even though a comment may be wrong from the omniscient perspective of the 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 opposition or the learning heuristic errs, the Advisors, again with the backing of more recent useful knowledge, can eventually override it through their comments. Hoyle, for example, has learned to imitate an expert's incorrect move in a particular situation, after much subsequent play deduced that it was an error, and then refused to repeat it.

Through its specification of general domain knowledge, FORR guarantees a minimal performance standard and offers graceful degradation. Hoyle, for example, can play any game in its domain correctly, if not expertly, given the game definition and its game-playing behavioral script. Even at a game it does not yet play well or at newly-encountered stronger opposition, Hoyle's moves reflect sensible underlying premises, its Advisors.

Finally, the clear distinction between control knowledge and declarative knowledge in FORR supports quick debugging and user-friendly visibility. Hoyle, for example, posts its comments on the screen as it plays, providing a clear indication of why it makes its choices.

FORR's principal limitations are reliance on human intelligence, reliance on experience, lack of creativity, and memory requirements. 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 are applicable to all tasks in the domain. The one-time effort to construct them is well invested, and FORR's modularity ensures that 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 has on what is learned, how quickly it is learned, and how useful that learned knowledge is (Epstein, 1992a). Hoyle's solution is to have the Learner deliberately provide alternative experiences 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 (Laird et al., 1987; Minton, 1988). 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. Hoyle does this, quite successfully, with whether or not it is an advantage to go first. Finally, 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.

When apprenticed to the kind of non-deterministic model Hoyle uses, learning is also non-deterministic. A program that learns from such a model will acquire 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 perfection achieved.

6. CONCLUSIONS

The right reasons for doing things, at least in the playing of some games, turn out to be relatively clear; it is the control structure for resolving conflict among those reasons that is complex. 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 which task. The tier hierarchy provides a commonsense foundation (for example, "if you remember this, do not calculate it") and then conflict resolution takes over.

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 problems susceptible to the same general problem-solving knowledge. Each problem requires a finite number of choices at a finite number of decision points. 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 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.

In two years of exploration with Hoyle, FORR has provided a surprisingly robust framework. Hoyle can learn even when its expert model is fallible; it has often learned to play better than a human-constructed expert model for a given game. The result is a collaborative development environment, a cycle where the expert model represents the best human knowledge about a game until Hoyle finds the flaws. For development in one complex and ill-understood skill domain, FORR works well. Other domains should now be explored.

ACKNOWLEDGMENTS

This work was supported in part by NSF 9001936 and PSC-CUNY 668287.

REFERENCES

Anderson, J.R. 1983. Acquisition of Proof Skills in Geometry. In *Machine Learning: An Artificial Intelligence Approach,* ed. R.S. Michalski, J.G. Carbonell and T.M. Mitchell. Palo Alto: Tioga Publishing.

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

Berlekamp, E.R., Conway, J.H. and Guy, R.K. 1982. *Winning Ways for Your Mathematical Plays.* 2 vols. London: Academic Press.

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*, 236-243. Morgan Kaufman.

Epstein, S.L. 1992b. Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems,* to appear.

Laird, J.E., Rosenbloom, P.S. and Newell, A. 1987. SOAR: An Architecture for General Intelligence.

Artificial Intelligence 33 (1): 1-64.

Michalski, R.S. 1983. A Theory and Methodology of Inductive Learning. In *Machine Learning: An Artificial Intelligence Approach,* ed. R.S. Michalski, J.G. Carbonell and T.M. Mitchell. Palo Alto: Tioga Publishing.

Minton, S. 1988. *Learning Search Control Knowledge - An Explanation-Based Approach.* Boston: Kluwer Academic.

Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M.N., and Schlimmer, J.C. 1990. Theo: A Framework for Self-Improving Systems. In *Architectures for Intelligence,* ed. K. Vanlehn. Boston: Erlbaum.

Mitchell, T.M. 1985. LEAP: A Learning Apprentice for VLSI Design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* 573-580. Morgan Kaufmann.

Steinberg, L. 1987. Design as Refinement Plus Constraint Propagation: The VEXED Experience. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* 830-835. Morgan Kaufmann.

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

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