

Learning through Interactive Behavior Specifications

Tolga Könik

Computational Learning Lab., CSLI
Stanford University
Stanford, CA 94305, USA
konik@stanford.edu

Douglas J. Pearson

ThreePenny Software, LLC
Seattle, WA 98103, USA
douglas.pearson@threepenny.net

John E. Laird

EECS Department
University of Michigan
Ann Arbor, MI 48109, USA
laird@umich.edu

Abstract

We describe a *learning from diagrammatic behavior specifications* approach, where the task-performance knowledge of a human expert is transferred to an agent program using abstract behavior scenarios that the expert and the agent program interactively specify. The diagrammatic interface serves as a communication medium between the expert and the agent program to share knowledge during behavior specification. A relational learning by observation component interprets these scenarios in the context of background knowledge and expert annotations to learn first-order rules that represent the task-performance knowledge for an improved agent program.

Introduction

Developing artificial cognitive agents that behave “intelligently” in complex environments (i.e. large, dynamic, nondeterministic, and with unobservable states) usually presumes costly agent-programmer effort for acquiring knowledge from experts and encoding it into an executable representation. We explore machine learning systems that automatize this process by transferring task-performance knowledge of a human user (“the expert”) to a agent program.

In this paper we describe a *learning from diagrammatic behavior specifications* approach, where the expert and the agent program interactively specify abstract behavior scenarios using a graphical interface that diagrammatically represents the task and the environment (Fig. 1). A learning component interprets these scenarios in the context of background knowledge, induces procedural rules consistent with the scenarios, and uses them to improve the agent program.

In many real-time tasks (e.g. car driving), only one agent can control the execution of the task at a time. In transferring knowledge from one agent to another in this setting, an important issue is how the “task-executer” and “observer” roles of the two agents are coordinated while the task is performed. One approach to solve this problem is to assign fixed roles to the expert and the agent program. For example, in behavioral cloning systems (Bratko,

Urbancic, and Sammut, 1998; Sammut et al. 1992), a learner agent passively observes the task execution of the expert. On the other hand, in learning by instruction systems (Huffman and Laird, 1995) an agent program executes the task while an observing expert gives instructions and feedback. Könik and Laird’s (2004) learning by observation framework support learning from behavior generated by both agents, but the agents do not generate behavior interactively.

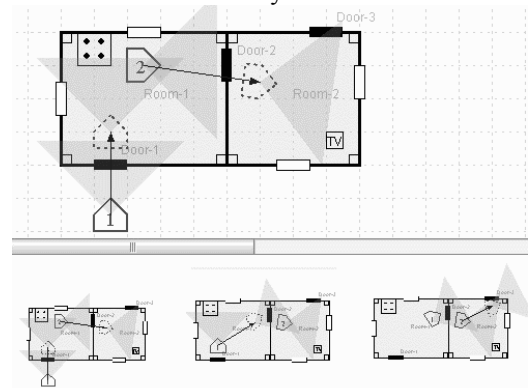


Fig. 1. Diagrammatic behavior specification with Redux

In contrast, in the interactive approach described in this paper, the agents switch roles dynamically depending on the specific situation, their knowledge about the world and about the other agent. We use the diagrammatic behavior specification tool Redux (Pearson and Laird, 2004) that facilitates this dynamic interaction by acting as a communication medium between the agents where they share assumptions about the world state, desired behavior, goals motivating that behavior, and feedback about each other’s behavior specifications. This interactive approach helps the learning system to focus on parts of the task where the agent program is lacking knowledge most.

A key bottleneck in creating an agent program by observing and replicating a human expert is that the decisions that the expert makes depend on more features than just those observable in the world. To overcome that problem, our diagrammatic representation provides a natural way for the expert to annotate behavior with

structures related to his/her mental reasoning, such as goals or markers highlighting objects relevant to a decision. Moreover, our framework allows inclusion of complex background knowledge about the task and the domain. These additional information sources help the learner to better model the processes internal to the expert. The learner first learns how the expert selects internal goals based on observed situations, active goals, and knowledge. Next, it learns how the expert selects actions that exhibit behavior consistent with those goals.

The situations that our agents encounter are inherently structured and often include complex relations between the objects. As the agents make decisions and generate behavior, they combine information about observed situations with complex internal structures such as goals and knowledge. We use a *relational learning by observation (ReLBO)* framework (König and Laird, 2004) that provides a natural way of combining these multiple complex knowledge sources by representing all available information using a first order language. ReLBO receives temporally changing relational situations as well as correct/incorrect actions and goal annotations as input, interprets them in the context of additional expert annotations and complex background knowledge, and finally induces an agent program that behaves similarly to the human expert. *ReLBO* reduces the “behave like an expert” learning problem, to a set of supervised learning problems that can be framed in an Inductive Logic Programming (ILP) setting (Lavrac and Dzeroski, 1994; Bratko and Muggleton, 1995), where first-order rules are learned from structured data.

Diagrammatic Behavior Specification Framework

In learning from diagrammatic behavior specifications setting depicted in Fig. 2, a human expert and an agent program interact through Redux, specifying behavior in a diagrammatic story-board like representation. The interaction starts with the expert actively specifying every step of the behavior. As the agent program improves through leaning, it becomes a more active participant, suggesting behavior and prompting the expert to resolve conflicts. The agent program takes greater control of the interaction until finally the expert is passively verifying its behavior.

The scenarios generated through the interaction of the expert and the agent program is fed to a relational learning by observation system, which interprets the annotated behavior in the context of background knowledge about the task and domain, and generates an improved agent program which replaces the current one. At each cycle, a new agent program is learned from scratch but since more behavior traces have been accumulated, a more accurate agent program is expected to be learned. When the expert decides that the agent program is competent enough, it is exported to an external agent architecture, currently Soar

(Laird and Rosenbloom, 1994; Laird, Newell, and Rosenbloom, 1987), which interacts with the real environment.

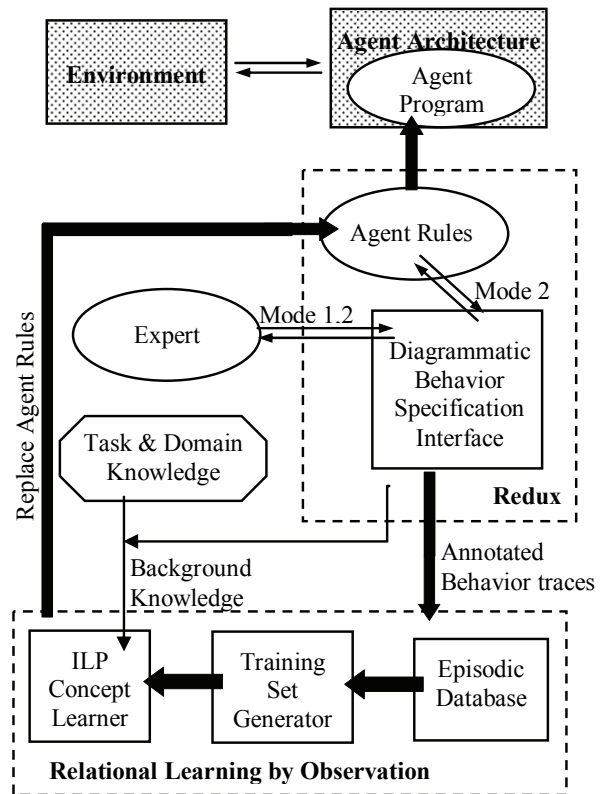


Fig. 2. Learning Framework

Representation of Task-Performance Knowledge

We assume that the task performance knowledge of the target agent program is decomposed into a hierarchy of operators that represent both the goals that the agents pursue and the actions that they take to achieve these goals (Fig. 3). The operators at the leaves represent primitive actions and the remaining operators represent the goals of the agent. With the operator hierarchy assumption, we decompose the “learning an agent program” problem to multiple “learning to manage the activity of an operator” problems. The suboperators correspond to strategies that the agent can use as part of achieving the goal of the parent operator. The agent has to continuously maintain the activity of these operators based on current sensors and internal knowledge. When the agent selects an operator, it must also instantiate its parameters. It then executes the operator by selecting and executing suboperators. The suboperators may be executed in complex orderings to achieve the goal of the parent operator, depending on the observed situations and internal knowledge. The real execution on the environment occurs when *actions*, the lowest level operators, are selected. Operators at any level can be retracted in response to the observed events

(changes in perception), in which case all of their suboperators are also retracted.

For example in Fig. 3, assume that the agent decides to get an item i_1 by selecting `get-item(Item)` and instantiating `Item = i_1`. If the agent is not in the same room with i_1 , it selects the suboperator `get-item-different-room(i_1)`. Then the agent executes this operator using its suboperators `go-to-door` and `go-through-door`. The operator `go-to-door` is used to select and approach a door leading towards the item i_1 . When the agent is close to that door, `go-to-door` is replaced with `go-through-door`, which moves the agent to the next room. Once in the next room, the agent selects `go-to-door` again but this time with a new Door instantiation. This process continues until agent is in the same room with i_1 and `get-item-different-room` is retracted with all of its suboperators and replaced with `get-item-in-room`.

We assume that there may be at most one operator active at each level of the hierarchy, except the lowest level operators, which are primitive actions that can be executed in parallel. This assumption simplifies the learning task because the learner associates the observed behavior only with the active operators and each operator is learned in the context of a single parent operator.

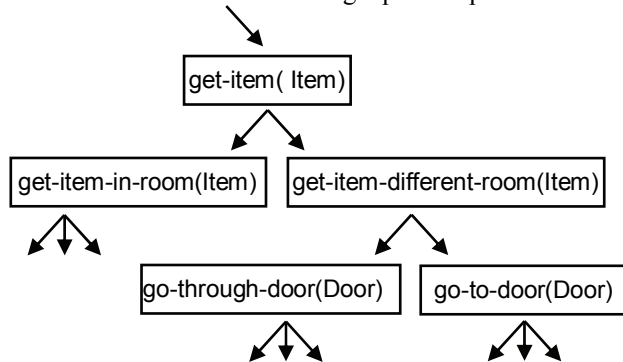


Fig. 3. An Operator hierarchy

The initial knowledge that the system has about the operators consists of their names and the number of arguments. The final agent obtained as the result of learning should have the capability of maintaining the activity of the operators (i.e. selecting them with correct parameters, stopping them when they achieve their goal, abandoning them in preference of other operators, etc.) and executing them (managing the suboperators).

Behavior Specification

The expert starts specification of a new behavior by drawing an initial situation consisting of an abstract representation of the environment. The expert then creates a scenario by selecting actions and generating new situations. The interface uses domain knowledge to help the expert with this process by predicting possible outcomes of the selected actions. For example, in a building navigation domain currently supported by Redux, the expert would first draw an abstract map representing

the building (rooms, doors, etc.), and select actions for “body” objects that represent the entities to be controlled by agent program at the end of the training. A scenario consists of discrete situations each representing important moments in a continuous behavior. The expert also annotates the scenario with goals and additional knowledge structures such as markers highlighting the important objects to communicate his/her reasoning to the agent program and the learning component.

If the agent program has some existing performance knowledge, the agent program aids the behavior specification of the expert by using that knowledge to determine what it would do in the current situation. Thus, it may suggest new operators. The expert validates, rejects or overrides these suggestions, which not only determines behavior in the current situation, but is also used by the learning component to improve the selection and termination of operators in the future. Since the behavior specification stage does not involve real-time behavior execution, the expert can move forward and backward in time, inspecting the collectively created scenario. Moreover, with the help of the agent program, the expert can consider multiple alternative operator selections in a situation, creating alternative history branches on the scenario. The expert can also specify undesired operator selections on the current branch of the scenario that should be avoided. These alternatives allow the learning system to get more information both about the variability and the limits of the correct behavior.

Relational Learning by Observation

The scenarios collaboratively generated by the expert and the agent program are recorded to an annotated behavior trace structure and returned to a *relational learning by observation* component, which interprets them in the context of background knowledge and induces a new agent program. During this process, the annotated behavior traces are first inserted into an episodic database (König and Laird, 2004) that efficiently stores and retrieves the observed behavior and the annotations. The training set generator maps the problem of “obtaining an agent program” to multiple problems of learning decision concepts that can be represented in a “supervised concept-learning” setting. The decision concepts include learning when the operators should be selected and when they should be terminated. For each decision concept, the training set generator creates positive and negative examples using the annotated behavior traces stored in the episodic database. The concept learner component uses an ILP algorithm that learns rules for each decision concept, by generalizing the examples in the training set in the context of background knowledge, which consists of a hand-coded domain theory and the annotated behavior traces. Finally, the learned rules are used to create a new

agent program, which interacts with the expert through Redux.

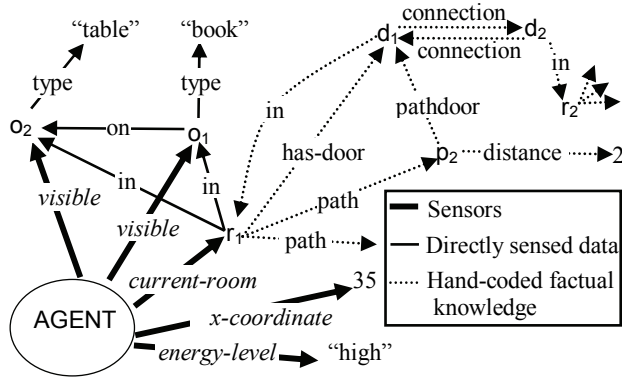


Fig. 4. An observed situation

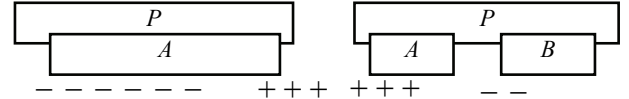
While the expert makes his/her decision based on the diagrammatic representation, the agent program and the learning system use a symbolic representation that directly corresponds to the diagrammatic representation. A *behavior trace* consists of a partially ordered set of situations, with each situation consisting of a set of binary relations symbolically representing the environment from the perspective of the agent program (Fig. 4). These relations can represent numerical sensors such as $x\text{-coordinate}(\text{agent}, 35)$ or $\text{energy-level}(\text{agent}, \text{high})$ as well as *object-valued* sensors such as $\text{current-room}(\text{agent}, r_1)$. The object valued sensors can be used to represent structured relations among perceived objects such as the “on” relation between the book object o_1 and the table object o_2 in Fig. 4. Moreover, they can represent factual knowledge assumed to be shared by the expert and the agent program. For example in Fig. 4, the learning system has not only the information that the agent is in the room r_1 , but also that the room r_1 is connected to the room r_2 through the door d_1 , although r_2 might not be directly visible. We say that the *observed situation predicate* $p(s_i, a, b)$ holds if and only if $p(a, b)$ holds at the situation s_i . The accepted and rejected annotations are represented with similar situation predicates (König and Laird, 2004).

Decision Concepts and Generating their Examples

Assuming an operator hierarchy, the problem of “learning an agent program” is decomposed into multiple “learning when an operator should be active” problems. We further decompose this problem into multiple “decision concept learning” problems that can be framed in a supervised ILP learning setting. Our current learning from behavior specifications implementation uses two decision concepts. The selection-condition concept describes when an operator should be selected and termination-condition described when an operator should be terminated. For example if $\text{selection-condition}(S, \text{go-to-door}(\text{Door}))$ holds for a situation $S=s_0$ and door object $\text{Door}=d_0$, it represents advice indicating that the agent should select the operator $\text{go-to-door}(d_0)$ at situation s_0 .

For a concept con and operator $\text{op}(x)$ where x is a parameter vector, the goal of learning is to return a first order concept of form: $\text{con}(s, \text{op}(x)) \leftarrow P(s, x)$, where P is a condition that can match the situation predicates in the episodic database and hand-coded background predicates. The positive and negative examples of decision concepts are ground terms of the form $\text{con}(s_0, \text{op}(x_0))$. The training set generator constructs these examples using the accepted and rejected goal annotations in the annotated behavior trace.

(a) termination-condition(op_A) (b) selection-condition(op_A)



(c) termination-condition(op_A) (d) selection-condition(op_A)

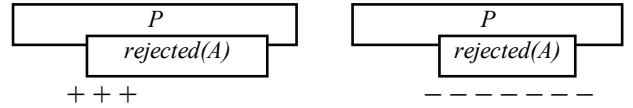


Fig. 5. Positive and negative example regions

The examples of a concept are created from a set of consecutive situations R called *positive (negative) example regions* such that for each $s_0 \in R$, $\text{con}(s_0, \text{op}(x_0))$ is a positive (negative) example. Fig. 5 depicts the positive and negative example regions of an operator op_A for two kind of decision concepts we consider in this paper. The horizontal direction represents consecutive situations in the behavior trace and the boxes represent the accepted annotation regions of three operators $\text{parent}(\text{op}_A)$, op_A , and op_B such that $\text{parent}(\text{op}_A)$ is the parent operator of op_A , and op_B is an arbitrary selected operator that shares the same parent with op_A . For example, we get the negative examples of the termination-condition concept of an operator op_A from a region of situations where op_A is selected but has not been terminated yet and we get its positive examples right after op_A is terminated (Fig 5.a). Similarly, the positive example region of the selection condition of op_A is where the expert has started pursuing op_A and its negative example region is where another operator is selected (Fig 5.b). If for example we have $\text{op}_A = \text{go-to-door}(d_1)$, $A = s_{20}\text{-}s_{30}$, and $B = s_{50}\text{-}s_{60}$, we could have the positive example selection-condition($s_{20}, \text{go-to-door}(d_1)$) and the negative example selection-condition($s_{50}, \text{go-to-door}(d_1)$). If behavior with negative goal annotations are available (either agent program specification rejected by the expert or undesired selection explicitly specified by the expert), these also provide examples for selection and termination conditions (Fig 5.c,d).

Learning Decision Concepts

The generated examples are given to an ILP algorithm, currently inverse entailment (Muggleton, 1995), to learn a theory that represents the decision concepts. The learning uses the examples generated by the training set generator and the background knowledge consisting of a hand-coded domain theory and the situation predicates stored in the episodic database. Like many ILP algorithms, inverse entailment conducts a heuristic search in the space of valid rules, evaluating the hypothesis for coverage of positive and negative examples. Fig. 6 depicts a correct hypothesis that is learned with this process. Note that we use Prolog syntax where the capitals are variables. It reads as: “At any situation *S* with an active goal *get-item(Item)*, the operator *go-to-door(Door)* should be selected if *Door* can be instantiated with a door in the room of the agent and on a path towards the *TargetRoom* which contains the *Item*.” Here, the learning system models the selection decision of *go-to-door* by checking the high-level goals and retrieving relevant information (active-goal retrieves information about the desired item), knowledge about the world (i.e. contains), and inferences (i.e. path). The path inference the agent makes depend on structured sensors such as current-room and knowledge about how the rooms are connected.

```
selection-condition(S, go-to-door(Door)) ←
    active-goal(S, get-item(Item) )      and
    agent-self(S, Agent)                 and
    path(S, Agent, Path)                 and
    pathdoor(S, Path, Door)              and
    destination(S, Path, TargetRoom)     and
    contains(S, TargetRoom, Item).
```

Fig. 6. A desired selection condition of *go-to-door*

The object valued parameters of the parent operator simplify the learning task, by providing access to the more relevant parts of the background knowledge by helping to find complex relations between the goals. For example in Fig. 6, the conditions for selecting the correct door could be very complex and indirect if the parent operator did not have the *Item* parameter that guides the search (i.e. the towards the room that contains the item).

Implementation

The relational learning by observation component is evaluated in (König and Laird, 2004). The current implementation of Redux allows the expert to specify behavior and mark decisions of the agent program as correct or incorrect. In the current implementation, the expert cannot dynamically change the agent generated behavior, but can achieve a similar effect by rejecting agent behavior and generating a new one.

We have only preliminary experimental results for the overall framework. In our initial experiments, the rule in Fig. 6 was learned in 2 learning cycles using a scenario

consisting of 10 situations. In the first cycle when the learner had only expert specified behavior, overgeneral rules were learned that select random doors in the room of the agent. In the second cycle, the expert gave feedback on incorrect selections of these rules and the learner was able to induce the correct rule in Fig. 6.

Related Work

Replicating the behavior of a human expert by passively observing is often called behavioral cloning. Most behavioral cloning research to date has focused on learning sub-cognitive skills in controlling a dynamic system such as pole balancing (Michie, Bain, and Hayes-Michie, 1990), controlling a simulated aircraft (Sammut et. al. 1992; Michie and Camacho, 1992), or operating a crane (Urbancic and Bratko, 1994). In contrast, our focus is capturing deliberate high-level reasoning, which we hope to achieve by interpreting the expert behavior in the context of additional information the expert provides and through an interaction with learned agent programs.

Using goals was proposed to improved robustness of the learned agents. Camacho’s system (1998) induced controllers that had goal parameters so that the execution system can use the same controllers under varying goal settings, but the goal settings are not learned. Isaac and Sammut (2003) present a two step approach where first a mapping from states to goal parameters is learned, then control actions are learned in terms of these goals. Suc and Bratko (2000) describe induction of qualitative constraints that model trajectories the expert is trying to follow to achieve goals.

In the goal-directed behavioral cloning research mentioned above, goals are desired values for some predefined parameters of a dynamic system. For example, the learning-to-fly domain has goal parameters such as target turn-rate. In contrast, the goals in our framework correspond to durative high-level internal states of the expert indicating that a particular kind of behavior is desired. Unlike the above approaches, we don’t assume pre-existing definitions for the goals. In our framework, goals are learned by learning when the experts select them as well as learning which behaviors become relevant once the goals are selected. The goals are hierarchically organized so that the goals at the high-levels of the hierarchy can represent complex behavior. van Lent’s (2000) learning by observation system KnoMic also learns hierarchies of durative goals using annotated behavior traces, although his goals don’t have parameters, preventing it from modeling complex relations between the goals. It uses an essentially propositional representation that limits its ability to deal with complex situations and to model complex reasoning of the human expert. None of these systems use behavior data generated though an interaction of the expert with an agent program. TRAIL (Benson and Nilsson, 1995) uses human behavior traces

together with experimentation in an ILP setting, but it learns what changes the actions cause, not to behave like the expert. Relational reinforcement learning with expert guidance (Driessens and Dzeroski, 2002) combine relational behavior traces generated by the experts with traces obtained from experimentation on the environment. Although their system use expert guidance to more quickly reach states that return reward, it doesn't use the choices of the experts as positive examples and learning is still based on linking the decisions to future reward signals. Neither of these systems try to replicate task performance behavior of the expert, which is our main focus in this paper.

Conclusion

Although the learning from diagrammatic behavior specifications approach described in this paper uses a learning by observation component, it addresses some of the important weaknesses of traditional learning by observation with real behavior performance data. The expert can use the diagrammatic interface to naturally express his/her mental reasoning, helping the learner to better model the expert. The diagrammatic interface provides a natural communication medium where the expert and the agent program share knowledge and specify behavior interactively, focusing to situations where the agent program is lacking knowledge most. Since the specified behavior is an abstraction of the real behavior, the learner can be biased to consider only the situations and objects that are relevant to a decision, and fewer examples may be sufficient to learn general knowledge.

Acknowledgment

This research was funded by a grant N61339-99-C-0104 from ONR.

References

Benson, S. and Nilsson, N. 1995. Inductive Learning of Reactive Action Models. In *Machine Learning: Proceedings of the Twelfth International Conference*, 47-54. Morgan Kaufmann.

Bratko, I. and Muggleton, S. 1995. Applications of Inductive Logic Programming. *Comm. of ACM* 38:65-70.

Bratko, I., Urbancic, T., and Sammut, C. 1998. Behavioural Cloning of Control Skills. In Michalski, R. S., Bratko, I., Kubat, M. (eds.). *Machine Learning and Data Mining : Methods and Applications*, 335-351. New York: J. Wiley.

Camacho, R. 1998. Inducing Models of Human Control Skills. In *10th European Conf. on Machine Learning*.

Driessens, K. and Dzeroski, S. 2002. Integrating Experimentation and Guidance in Relational Reinforcement Learning. In *Proceedings of the Nineteenth*

International Conference on Machine Learning (ICML-2002), 115-122.

Huffman, S. B. and Laird, J. 1995. Flexibly Instructable Agents. *Journal of Artificial Intell. Research* 3:271-324.

Isaac, A. and Sammut, C. 2003. Goal-directed Learning to Fly. In *Proceedings of the Twentieth International Conference (ICML 2003)*.

Könik, T. and Laird, J. 2004. Learning Goal Hierarchies from Structured Observations and Expert Annotations. In *14th International Conference on Inductive Logic Programming, Lecture Notes in AI, Vol. 3194*. Springer.

Laird, J. E., Newell, A., and Rosenbloom, P. S. 1987. Soar: An Architecture for General Intelligence. *Artificial Intelligence* 33: 1-64.

Laird, J. and Rosenbloom, P. S. 1994. the Evolution of the Soar Architecture.

Lavrac, N. and Dzeroski, S. 1994. Inductive Logic Programming Techniques and Applications. New York: Ellis Horwood.

Michie, D., Bain, M., and Hayes-Michie, J. 1990. Cognitive Models from Subcognitive Skills. In *Knowledge-Based Systems in Industrial Control.*, 71-90. Stevenage: Peter Peregrinus.

Michie, D. and Camacho, R. 1992. Building Symbolic Representations of Intuitive Real-Time Skills From Performance Data. In *Machine Intelligence 13 Workshop*.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Generation Computing* 13:245-286.

Pearson, D. J. and Laird, J. E. 2004. Redux: Example-Driven Diagrammatic Tools for Rapid Knowledge Acquisition. In *Conference on Behavior Representation in Modeling and Simulation*.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. 1992. Learning to fly. In *Proceedings of the 9th International Conference on Machine Learning (ICML-1992)*, 385-393.

Suc, D. and Bratko, I. 2000. Problem decomposition for behavioural cloning. In *11th European Conference on Machine Learning. Lecture Notes in Computer Science, Vol. 1810.*, 382-391. Germany: Springer-Verlag.

Urbancic, T. and Bratko, I. 1994. Reconstructing Human Skill with Machine Learning. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, 498-502. John Wiley and Sons.

van Lent, M. 2000. Learning Task-Performance Knowledge Through Observation. Ph.D. diss., Univ. of Michigan.