Learning procedures from interactive natural language instructions*

Scott B. Huffman and **John E. Laird**
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, Michigan 48109-2110
huffman@umich.edu

Abstract

Despite its ubiquity in human learning, very little work has been done in artificial intelligence on agents that learn from interactive natural language instructions. In this paper, we examine the problem of learning procedures from interactive, situated instruction, in which the student is attempting to perform tasks within the instructional domain, and asks for instruction when it is needed. We present Instructo-Soar, a system that behaves and learns in response to interactive natural language instructions. Instructo-Soar learns completely new procedures from sequences of instruction, and also learns how to extend its knowledge of previously known procedures to new situations. These learning tasks require both inductive and analytic learning. Instructo-Soar exhibits a multiple execution learning process in which initial learning has a rote, episodic flavor, and later executions allow the initially learned knowledge to be generalized properly.

1 Introduction

The hallmark of universal computation systems is their ability to take instructions. This ability separates computers from other machines, which can perform only a limited number of tasks. Instructability allows computers to perform any of an infinite number of tasks. However, computers take instructions in a way that is radically different from the way humans do. Computers receive instructions in the form of programs. This method of communication from instructor (programmer) to computer is characterized by the following properties.

- **Artificial language.** Programmers must translate knowledge into a language that requires precise artificial terminology and syntax.
- **Unsituated instruction.** The instruction does not occur within the context of the computer attempting to solve a specific problem.
- **Non-interactive instruction.** The instructor determines when and what to instruct without any feedback from the computer.

These properties have a number of implications for the instructor:

1. The instructor must know what procedures are already encoded in the computer, to avoid redundancy and conflicts.
2. The instructor must understand the effects of long sequences of instruction, because a complete instructional sequence must be generated.
3. The instructor must create a procedure that applies in every situation it will be exposed to.
4. The instructor must specify the procedures in complete detail; no steps may be omitted or abstracted.

*This paper also appeared in *Machine Learning: Proceedings of the Tenth International Conference*, ed. Paul E. Utgoff, Amhearst, Mass., June 1993.

In contrast, humans can engage in apprenticeship instruction, in which the student actively tries to acquire knowledge to aid in problem solving. This type of instruction has the following properties:

- **Natural language.** Instructor and student speak the same language, and the language is highly flexible.
- **Situated instruction.** The instructor and student are situated within a specific task. The instructor does not need to predict the effects of long instruction sequences because the student performs the task in response to individual instructions. The instructor needs to produce instructions for only the situation at hand, not for all possible situations. The student can use the situation to disambiguate instruction, and cue the recall of relevant domain knowledge.
- **Interactive instruction.** The student can request help during a task. This frees the instructor from specifying the procedure in full detail; instructions are given when needed. The instructor need not know exactly what the student knows about the task. If the student is unable to fill in missing steps or details, more instruction can be requested.

In this paper, we describe Instructo-Soar, a system that learns procedures from interactive natural language instructions. Instructo-Soar attempts to solve problems within a task domain, and requests instruction when it does not know how to make progress on a problem. The instructions are simple English imperative sentences. They can include commands for primitive or known operators, for complex operators that the system does not know how to apply in the current situation, and for completely new complex operators that the system must learn from scratch. These latter cases lead to a recursive use of instruction where the system learns a hierarchy of operators. Both analytic and empirical learning methods are employed so that after instruction, the system can perform similar tasks (and subtasks) without instruction. Learning of a new procedure is initially by rote, using an episodic memory acquired as a side-effect of natural language comprehension. During later executions of the task, analytic techniques generalize the procedure.

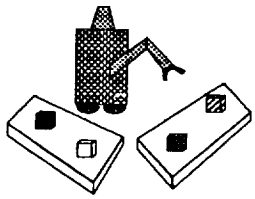
2 Related Work

This work is most closely tied to work on learning from external guidance and advice taking [McCarthy, 1968; Carbonell *et al.*, 1983]. Prior research in these areas has usually emphasized one of the following: natural language instruction, situated instruction, or interactive instruction.

SHRDLU [Winograd, 1972] learned new goal specifications by directly transforming sentences into state descriptions, but did not learn how to perform procedures. Others have learned declarative knowledge bases from natural language (e.g., [Haas and Hendrix, 1983]). A number of recent systems perform in response to natural language input, but do no learning [Vere and Bickmore, 1990; Chapman, 1990; DiEugenio and White, 1992]. Lewis *et al.* [1989] present a system that learns operator sequences from natural language instructions taken in batch form (unsituated and non-interactive). Alterman *et al.* [1991] and Martin and Firby [1991] describe situated systems that recover from execution errors by learning from instruction.

There have been a variety of systems that learn from observation [Redmond, 1989; Segre, 1987; Wilkins, 1988; VanLehn, 1987]. These systems take traces of expert behavior on a specific problem and learn general procedures using analytic techniques such as explanation-based learning [Mitchell *et al.*, 1986]. However, these system do not support interaction with the instructor.

Learning apprentice systems (LAS's) [Mitchell *et al.*, 1990; Kodratoff and Tecuci, 1987; Golding *et al.*, 1987; Laird *et al.*, 1990] extend the work on learning from observation by providing some interactivity: typically, the system suggests steps to the instructor. However, the instructor can only select actions that are directly performable at the current problem state. LAS's cannot take instructions specifying new, unknown actions (that thus must be learned) or actions with unmet preconditions (which the agent may or may not know how to achieve). For example, LEAP's [Mitchell *et al.*, 1990] instructor inputs a complete circuit implementing a desired function, but cannot instruct the system to perform some new, unknown circuit transformation. Since whole circuits are learned for each function, LEAP avoids the problem of an instructor "skipping steps" by specifying operations with unmet preconditions. In addition, learning apprentice systems require that either the termination conditions of the procedure being taught (the goal concept [Mitchell *et al.*, 1986]) are already known, or that the instructor provide a complete description of termination conditions. Finally, these systems typically have no natural language capability.



(a)

Pick up the red block.
Move to the yellow table.
Move the arm above the red block.
Move up.
Move down.
Close the hand.
Move up.
The operator is finished.

(b)

Figure 1: (a). Initial situation of agent; (b) Instructions to teach operator.

3 Instruction within an autonomous agent

One factor that most previous work on instruction has ignored is the integration of learning from instruction within an autonomous agent. To learn from interactive instruction, an autonomous agent must have general reasoning capabilities, and be able to recognize when its knowledge is insufficient and instruction is needed. Instructions must be assimilated into a possibly large body of existing knowledge, and instructional learning must be smoothly integrated with the agent's other learning and problem solving methods. An agent must maintain its ability to respond to its environment even while accepting instructions, and must be able to apply learning from instruction to a wide variety of tasks.

Supporting these capabilities is dependent in part upon the architecture in which the agent in constructed. We use Soar [Laird *et al.*, 1987] as our underlying architecture. Soar's basic structure provides a framework in which these capabilities can be approached.

In Soar, all activity occurs by applying operators to states within problem spaces, supporting general problem solving and planning. Our instruction learning techniques learn and extend operators, and thus have the potential to be applicable to any problem encoded in Soar. When a Soar agent cannot make progress within a problem space, an impasse arises, and a subgoal is generated to resolve the impasse. Any type of knowledge might be applied within the subgoal, so learning from instruction can co-exist with learning from other knowledge sources, such as experimentation, analogy, etc. Learning occurs through chunking, a form of explanation-based learning, which summarizes the results of subgoal processing, avoiding the impasse in the future. Chunking occurs over all subgoals, so instructional learning can be performed as part of the ongoing activity of the agent, rather than using a separate mechanism that interrupts the normal course of activity.

Instructo-Soar's problem spaces implement an agent with three main categories of knowledge: natural language processing knowledge, originally developed for NL-Soar [Lehman *et al.*, 1991]; knowledge about obtaining and using instruction; and knowledge of the task domain itself. This task knowledge is extended through learning from instruction. Assumed characteristics of the Instructo-Soar agent include:

1. **Relevant relationships.** The agent has knowledge of all of the relevant task relationships, and can derive them from perception.
2. **Primitive operators.** The agent knows a set of primitive operators that it can execute, internally simulate, and map natural language to.
3. **Reading ability.** The agent can read the instructions, even if it has no knowledge of an operation within the current task domain that corresponds to the instruction.
4. **Locality of instruction.** The agent assumes that an instruction applies to the *most recent* unachieved operation.

4 Learning from instruction

Consider the agent and situation shown in Figure 1(a). The agent has primitive operators for moving to tables, opening and closing its hand, and moving its arm up, down, and into relationships with objects (e.g., above blocks). This is the primary domain Instructo-Soar has been applied to; the techniques have also been applied in a more limited way to a flight domain, in which Soar controls a flight simulator and instructions are given for simple maneuvers like taking off [Pearson *et al.*, 1993]. To explain Instructo-Soar's performance, we will use the example of teaching the agent in Figure 1(a) to pick up blocks. Since picking up blocks is not a known operator, when told "Pick up the red block," the agent must learn a new operator.

To learn a new operator, an agent must learn each of the following:

1. **Mapping from natural language:** What instruction(s) map onto the new operator. The mapping allows the agent to select the operator when commanded in the future.
2. **Operator template:** Knowledge of the operator's arguments and how they can be instantiated. For picking up blocks, the agent acquires a new operator with a single argument, which may be instantiated with any block that isn't currently picked up.
3. **Implementation:** How to perform the operator. New operators are built from primitive and/or previously learned operators, so implementation takes the form of a series of sub-operators (e.g., move to the proper table, grasp the block, etc.)
4. **Termination:** Knowledge of when the new operator is achieved. This is the goal concept of the new operator. For "pick up", the termination conditions include holding the desired block, with the arm up.

Instructo-Soar handles simple imperative sentences, and learns a straightforward mapping of an instruction's semantic argument structure to a newly generated operator template. In general, mapping from instructions to task operators and objects can be difficult, as it can require complex natural language comprehension, and possibly reasoning about the task itself [Huffman and Laird, 1992; DiEugenio, 1992].

To learn a general operator implementation from instructions, an agent must determine the proper scope of applicability of each instruction. Some features of the current task and situation are important conditions, while others may be ignored. For example, when told to pick up a red block, does it matter that the block is red? Perhaps, if building a stoplight. But if trying to block open a door, the key feature may be the block's weight. Thus, learning from instruction can involve both generalization (that "red" doesn't matter, although explicitly mentioned) and specialization (that the weight matters, although not mentioned).

To learn general implementations, Instructo-Soar uses explanation-based learning (EBL) as realized by chunking in Soar. Proper generalization requires understanding how each instruction contributes to achieving the goal. However, during the initial execution of the instructions for a new operator, the agent does not know the termination conditions (goal concept) of the operator; therefore, generalization on this basis is impossible. Thus, initial learning is based on a weak inductive step: believing what the instructor says. This learning is rote and overspecific, with an "episodic" flavor. At the end of the initial execution, the termination conditions, or goal concept, of the new operator can be induced. On later executions, the agent *can* form an understanding of how the instructions, recalled from its episodic memory, allow the goal to be reached, using its knowledge of primitive operator effects (domain theory). This allows the implementation sequence to be learned deductively (and generally), based on achievement of the induced goal concept. We will describe the details of the technique using an example.

5 Example

Consider the agent shown in Figure 1(a) being instructed to pick up blocks. The agent is given the instructions shown in Figure 1(b) during the course of performing the task.

5.1 First execution

The agent begins with knowledge of the primitive operators, but no knowledge of the new operator it will be instructed to perform. Following the first execution, the agent must be able to perform at least the exact same task without being re-instructed. Thus, the agent must learn, in some form, *all* of the parts of the new operator, as described in the previous section.

The first instruction given is "Pick up the red block." It is comprehended using Soar's natural language capability, NL-Soar [Lehman *et al.*, 1991], which produces a semantic structure and resolves "the red block" to a block in the agent's environment. However, the semantic structure produced does not correspond to any known operator, indicating that the agent must learn a new operator. Thus, a new, empty operator is generated (e.g., `new-op14`), with an argument structure that directly corresponds to the semantic structure's arguments (here, one argument, `object`). The system learns a mapping from the semantic structure to the new operator, heuristically restricting arguments to be of the same type (e.g., `isa block`) as in the current instruction.

Next, the new operator is selected for execution. Since the agent doesn't know any implementation for the operator, it immediately impasses and asks for further instructions. Each instruction in Figure 1(b)

is given, comprehended and executed in turn. For instance, "Move to the yellow table" is comprehended, mapped to a known operator, and executed. These instructions provide the implementation for the new operator.

The instruction "Move the arm above the red block" provides an example of learning to achieve the preconditions of a known operator. This operation is known, and the agent can perform it when its hand is in the upper plane. However, here the hand is in the lower plane, so the operation cannot be performed directly. Thus, the agent asks for instruction about this operator, and is told to move the arm up. This achieves the precondition, and after moving up the agent has sufficient knowledge to complete the move above operation without further instruction. As a result, a rule is learned that will propose moving up to achieve this precondition in the future. Thus, a known operation has been extended to apply in a new situation; further instruction could extend it even further, for instance allowing the agent to "move above" starting from a state where it's not even next to the table. Note that the interactive nature of instruction means that the instructor *need not know* beforehand whether the agent knows how to apply an operation from the current situation. This recursive instruction of sub-operations could be multiple levels deep, allowing for a great flexibility of instruction sequences, depending what the agent already knows. A simple mathematical analysis shows that for a sequence of only six primitive actions, with preconditions for each action, over 100 possible sequences of interactive instruction are possible [Huffman, 1992]. This contrasts with learning from observation, in which systems learn from observing only the sequence of primitive operations performed to carry out the task.

After completing the "move above" action, the agent continues receiving and executing instructions for the new "pick up" operator. Ultimately, the implementation sequence for "pick up" will be learned at the proper level of generality, based on understanding how each instructed operator leads to successful execution of the new operator. During the initial execution, however, this is impossible, because the goal of this new operator is not yet known. Thus, the agent resorts to *rote learning*, recording exactly what it was told to do, in exactly what situation.

This recording process is not an explicit memorization step; rather, it occurs as a side effect of language comprehension. While reading each sentence, the agent learns a set of rules that encode the sentence's semantic features. The rules help NL-Soar to resolve referents in later sentences (implementing a simple version of Grosz's focus space mechanism [Grosz, 1977]). The rules record each instruction, indexed by the goal it applies to and its place in the instruction sequence. This episodic "case" corresponds to a lock-step, overspecific sequencing of the instructions given to perform the new operator. For instance, the agent encodes that "to pick-up (that is, new-op14) the red block, rb1, I was first told to move to the yellow table, yt1." One issue that arises here is whether and when to generalize the index and information contained within the case. However, at this point any generalization would be purely heuristic, since the agent was unable to explain the various steps of the episode.

Finally, the agent is told "The operator is finished," indicating that the goal of the new operator has been achieved. This triggers the agent to learn termination conditions for the new operator. Learning termination conditions is an inductive concept formation problem. Standard concept learning approaches may be used here; however, typically, an instructor will expect learning within a small number of examples. Currently, the system uses a simple heuristic: it compares the current state to the initial state the agent was in when commanded to perform the new operator. Everything that has *changed* is considered a part of the termination conditions of the new operator. In this case, the changes are that the robot is standing at a table, holding a block, and the block and gripper are both up in the air.

This heuristic forms the system's inductive bias for learning termination conditions. It allows learning from a single example, but is clearly too simple. Conditions that changed may not matter; e.g., perhaps it doesn't matter to picking up blocks that the robot ends up at a table. Unchanged conditions may matter; e.g., if learning to build a "stoplight", block colors are important.

Instructo-Soar performs this induction by EBL over an overgeneral theory (as, e.g., [Miller and Laird, 1991; VanLehn *et al.*, 1990; Rosenbloom and Aasman, 1990]). Although not sophisticated here, this type of inductive learning has the advantage that the agent can alter the bias to reflect other available knowledge. This might include more instruction (e.g., simply asking which features are relevant [Laird *et al.*, 1990]); analogy to other known operators (e.g., pick up actions in related domains), domain heuristics, etc.

On the first pass, then, the agent:

- Carries out a sequence of instructions achieving a new operator.
- Learns a new operator template.

- Learns the mapping from natural language to the new operator.
- Learns a rote execution sequence for the new operator.
- Induces the termination conditions of the new operator.

Since the agent has learned all of the necessary parts of an operator, it will be able to perform the same task again without instruction. However, since the implementation of the operator is rote, it can only perform the *exact* same task. It has not learned generally how to pick up blocks yet.

Since the goal is now known, the system could explain and generalize the instruction sequence directly after the first execution. This is a reasonable possibility, but the multi-step simulation required has two disadvantages:

1. The agent's ongoing performance of the tasks at hand (either by acting or by taking more instruction) is temporarily suspended. This could be awkward if instruction of the new procedure being simulated is nested within the instructions for larger tasks that must still be completed, or if these tasks have temporal constraints.

2. The multiple step simulation is susceptible to compounding of domain theory errors. That is, a significant error in simulating any step of the procedure (or the interaction of multiple small errors) can lead to an incomplete or incorrect explanation of goal achievement. Simulating to explain each ***individual*** instruction, as described below, avoids this problem because each successive simulation begins from the current external state, which reflects the true effects of the previous instructions.

Thus, we have opted for generalizing on future executions.

5.2 Later executions

Later, in the same situation the agent is again asked to pick up the red block. The agent selects the newly learned operator, and then reaches an impasse because it does not yet know the general implementation sequence for the operator (how to pick up blocks in the general case). Here, the agent attempts to recall instructions it was given during the first execution. It retrieves, instruction by instruction, the rote case it learned previously.

After each retrieval, before carrying out the instruction in the external world, the agent attempts to explain to itself *why* the instructed operator leads to achievement of the higher-level goal of picking up the block. This explanation attempt takes the form of an internal simulation. Starting from the current world state, the agent internally simulates the recalled operator. Thus, the *situatedness* of the instruction plays a key role in the learning process, because the current situation grounds the explanation. The agent continues to simulate operators until it either reaches its higher-level goal (internally) or does not know what to do next. If the goal is reached, the path taken to the goal comprises an explanation of how the recalled operator leads to goal achievement.

From this explanation, the system learns a general rule that proposes the recalled operator under the right conditions. The rule both generalizes and specializes the original instruction. In "move to the yellow table"'s case, the color of the table is generalized away, because it was not critical for achievement of the goal, while the fact that the table has the block to be picked up on it is included in the proposal rule's conditions.

After learning the complete general implementation, the agent will perform the task without reference to the rote case. If asked to "Pick up the green block," `new-op14` is selected and instantiated with the green block as its argument. Then, the general sub-operator proposal rules for `new-op14` fire one by one, implementing the operator, until finally the termination conditions are recognized and the operator is terminated. Since the general proposal rules test state conditions directly, the agent can perform the task starting from any state along the implementation path, and can react to unexpected conditions (e.g., another robot stealing the block). In contrast, the rote implementation had to be performed from the same initial state each time, and its steps were not conditional on the state.

6 Results

In the robotic domain described earlier, Instructo-Soar has been applied to learn a hierarchy of task operators, shown in Figure 2. The system read 24 natural language instructions (14 unique sentences) and learned 1357

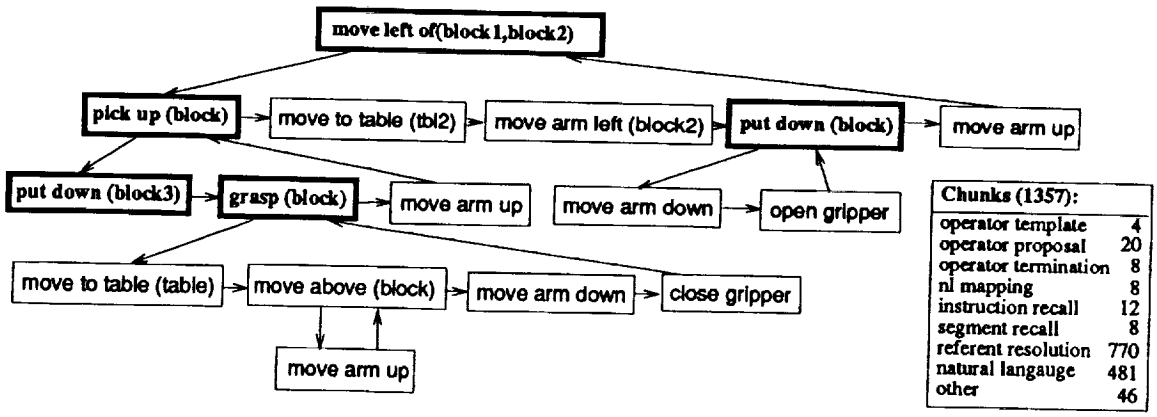


Figure 2: A hierarchy of operators learned by Instructo-Soar. Primitive operators are shown in light print; learned operators are in boldface.

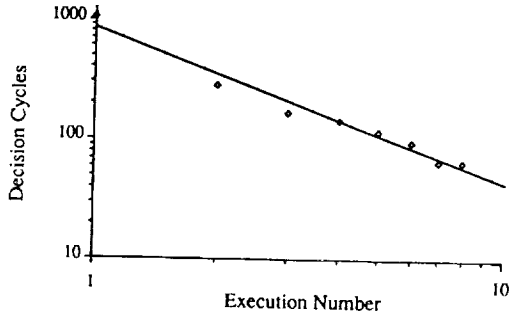


Figure 3: Decision cycles vs. learning trial for executing "pick up".

chunks broken down as shown in the figure. It learned four completely new operators (shown in bold), how to achieve preconditions for a known operator (`move above`), and the extension of a new operator (extending "pick up" to work if the robot already is holding a block after initially learning "pick up" when the gripper was empty). This hierarchy can be taught using many different instruction orderings. For instance, new operators that appear as sub-operators (e.g., `grasp`) can be taught either before teaching higher operators (e.g., `pick up`), or during teaching of them. If during, the agent recursively requests instruction for the lower new operator. Thus the instructor need not know whether the agent knows a procedure before commanding it. This is an important advantage of interactive instruction for autonomous agents, which may have large amounts of knowledge. Similarly, the instructor may suggest an action that has unmet preconditions (thus skipping steps in the instruction sequence), assuming the agent knows how to achieve them before performing the action. If the agent does not, it can request more instruction, and learn how to achieve the preconditions.

Instructo-Soar exhibits a number of interesting learning characteristics:

- **Multiple recall strategies.** Instructo-Soar has two strategies it can use in recalling past instructions. After recalling and internally simulating an instructed operator, the agent still may not know how that operator leads to the goal. At this point, the agent may terminate its internal simulation, and carry out the recalled operator in the external world. This is a *single recall* strategy, which is appropriate when the agent is under pressure to act quickly. Alternatively, the agent may attempt to recall further instructions, simulating each in turn, until the higher-level goal is reached. This is a *multiple recall* strategy, which leads to faster learning, but is more susceptible to errors in the agent's domain theory (primitive operator knowledge), as described above.
- **Bottom-up learning** (single recall strategy). Limiting recall to a single step allows only a single sub-operator per execution to be generalized. Generalized learning begins at the end of the implementation sequence and moves towards the beginning. As Figure 3 shows for learning "pick up", the resulting learning curve closely approximates the power law of practice [Rosenbloom and Newell, 1986] ($r = 0.98$).

C-5.

- **Effectiveness of hierarchical instruction** (single recall strategy). Due to the bottom-up effect, the system learns more quickly when taught hierarchical organizations than flat sequences. General learning for an N step operator takes N executions using a flat instruction sequence. Taught hierarchically as $\sqrt{N}$ sub-operators with $\sqrt{N}$ steps each, only $\sqrt{N}$ executions are required for full general learning. Hierarchical organization has the additional advantage that more operators are learned that can be used in future instruction.
- **Degradation without domain theory**. If the agent does not have knowledge of the primitive operators' effects, learning degrades to rote learning. This appears to be consistent with psychological research showing that subjects given procedural instructions learn and perform better when they have a domain model [Kieras and Bovair, 1984].

7 Conclusion

We have described Instructo-Soar, an agent that learns and extends procedures by receiving interactive, situated natural language instructions. The agent learns completely new operators: preconditions, implementation, and termination conditions (goal concept), in contrast to learning apprentice systems, which learn only implementations, and preconditions of those implementations, for known operators. New operators learned by Instructo-Soar may be specified in later instructions for other operators, leading to learning of operator hierarchies. From the initial execution of a new operator, the agent learns a rote, overspecific execution sequence, and induces termination conditions. On later executions, the execution sequence is generalized by using internal simulation to explain each instruction.

Instructo-Soar can be extended in a number of directions. In addition to positive imperative sentences, we are currently investigating learning from other types of instructions, such as positive and negative constraints, conditionals, and actions with monitoring conditions [Huffman and Laird, 1992]. The difference-of-states method used to induce operator termination conditions is being extended to allow instructor feedback about the induced conditions. Finally, allowing weaker forms of explanation, such as analogy and heuristic causality theories (e.g., [Pazzani, 1991; VanLehn *et al.*, 1992; Schank and Leake, 1989]), would lead to more graded degradation of learning with domain theory incompleteness. These types of explanation might also lead the agent to alter its basic domain theory, for instance by inferring previously unknown affects of primitive actions.

Acknowledgments

This research was sponsored by NASA/ONR under contract NCC 2-517. The authors wish to thank Paul Rosenbloom, Jill Lehman, Rick Lewis, and Randy Jones for valuable comments on earlier drafts.

References

[Alterman *et al.*, 1991] Richard Alterman, Roland Zito-Wolf, and Tamitha Carpenter. Interaction, comprehension, and instruction usage. Technical Report CS-91-161, Computer Science Department, Brandeis University, July 1991.

[Carbonell *et al.*, 1983] Jaime G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach*. Morgan Kaufmann, 1983.

[Chapman, 1990] David Chapman. *Vision, Instruction, and Action*. PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, April 1990.

[DiEugenio and White, 1992] B. DiEugenio and M. White. On the interpretation of natural language instructions. In *Proceedings COLING 92*, July 1992.

[DiEugenio, 1992] B. DiEugenio. Understanding natural language instructions: The case of purpose clauses. In *Proceedings of Annual Meeting of the ACL*, July 1992.

[Golding *et al.*, 1987] A. Golding, P. S. Rosenbloom, and J. E. Laird. Learning search control from outside guidance. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 334–337, August 1987.

[Grosz, 1977] Barabara J. Grosz. *The Representation and use of focus in dialogue understanding.* PhD thesis, University of California, Berkeley, 1977.

[Haas and Hendrix, 1983] Norman Haas and Gary G. Hendrix. Learning by being told: Acquiring knowledge for information management. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach.* Morgan Kaufmann, 1983.

[Huffman and Laird, 1992] Scott B. Huffman and John E. Laird. Dimensions of complexity in learning from interactive instruction. In Jon Erikson, editor, *Proceedings of Cooperative Intelligent Robotics in Space III, SPIE Volume 1829,* November 1992.

[Huffman, 1992] Scott B. Huffman. An analysis of instruction sequences. Artificial Intelligence Laboratory, University of Michigan, 1992.

[Kieras and Bovair, 1984] David E. Kieras and Susan Bovair. The role of a mental model in learning to operate a device. *Cognitive Science,* 8:255–273, 1984.

[Kodratoff and Tecuci, 1987] Yves Kodratoff and Gheorghe Tecuci. DISCIPLE-1: Interactive apprentice system in weak theory fields. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* pages 271–273, August 1987.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence,* 33(1):1–64, 1987.

[Laird *et al.*, 1990] John E. Laird, Michael Hucka, Eric S. Yager, and Christopher M. Tuck. Correcting and extending domain knowledge using outside guidance. In *Proceedings of the Seventh International Conference on Machine Learning,* 1990.

[Lehman *et al.*, 1991] Jill Fain Lehman, Richard L. Lewis, and Allen Newell. Natural language comprehension in Soar: Spring 1991. Technical Report CMU-CS-91-117, School of Computer Science, Carnegie Mellon University, March 1991.

[Lewis *et al.*, 1989] Richard L. Lewis, Allen Newell, and Thad A. Polk. Toward a Soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Annual Conference of the Cognitive Science Society,* August 1989.

[Martin and Firby, 1991] Charles E. Martin and R. James Firby. Generating natural language expectations from a reactive execution system. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society,* pages 811–815, August 1991.

[McCarthy, 1968] J. McCarthy. The advice taker. In M. Minsky, editor, *Semantic Information Processing,* pages 403–410. MIT Press, 1968.

[Miller and Laird, 1991] C. Miller and J. E. Laird. A constraint-motivated lexical acquisition model. In *Proceedings of the 13th Annual Conference on the Cognitive Science Society,* pages 827–831, Boston, 1991.

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

[Mitchell *et al.*, 1990] T. M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. LEAP: A learning apprentice system for VLSI design. In Yves Kodratoff and R. S. Michalski, editors, *Machine Learning: An artificial intelligence approach, Vol. III.* Morgan Kaufmann, 1990.

[Pazzani, 1991] Michael Pazzani. Learning to predict and explain: An integration of similarity-based, theory driven, and explanation-based learning. *Journal of the Learning Sciences,* 1(2):153–199, 1991.

[Pearson *et al.*, 1993] Douglas J. Pearson, Scott B. Huffman, Mark B. Willis, John E. Laird, and Randolph M. Jones. Intelligent multi-level control in a highly reactive domain. In *Proceedings of the International Conference on Intelligent Autonomous Systems,* Pittsburgh, PA., February 1993.

[Redmond, 1989] Michael Redmond. Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the International Workshop on Machine Learning,* pages 20–22, 1989.

[Rosenbloom and Aasman, 1990] Paul S. Rosenbloom and Jans Aasman. Knowledge level and inductive uses of chunking (ebl). In *Proceedings of the National Conference on Artificial Intelligence,* 1990.

[Rosenbloom and Newell, 1986] Paul S. Rosenbloom and Allen Newell. The chunking of goal hierarchies: A generalized model of practice. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach, Volume II.* Morgan Kaufmann, 1986.

[Schank and Leake, 1989] Roger C. Schank and David B. Leake. Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40:353–385, 1989.

[Segre, 1987] Alberto Maria Segre. A learning apprentice system for mechanical assembly. In *Third IEEE Conference on Artificial Intelligence for Applications*, pages 112–117, 1987.

[VanLehn *et al.*, 1990] K. VanLehn, W. Ball, and B. Kowalski. Explanation-based learning of correctness: Towards a model of the self-explanation effect. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, pages 717–724, 1990.

[VanLehn *et al.*, 1992] Kurt VanLehn, Randolph M. Jones, and Michelene T. H. Chi. A model of the self-explanation effect. *Journal of the learning sciences*, 2(1):1–59, 1992.

[VanLehn, 1987] Kurt VanLehn. Learning one subprocedure per lesson. *Artificial Intelligence*, 31(1):1–40, 1987.

[Vere and Bickmore, 1990] Steven Vere and Timothy Bickmore. A basic agent. *Computational Intelligence*, 6:41–60, 1990.

[Wilkins, 1988] David C. Wilkins. Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the National Conference on Artificial Intelligence*, pages 646–651, 1988.

[Winograd, 1972] Terry Winograd. *Understanding Natural Language.* Academic Press, New York, 1972.