Learning from highly flexible tutorial instruction*

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

Abstract

Situated, interactive tutorial instructions give flexibility in teaching tasks, by allowing communication of a variety of types of knowledge in a variety of situations. To exploit this flexibility, however, an instructable agent must be able to *learn* different types of knowledge from different instructional interactions. This paper presents an approach to learning from flexible tutorial instruction, called *situated explanation*, that takes advantage of constraints in different instructional contexts to guide the learning process. This makes it applicable to a wide range of instructional interactions. The theory is implemented in an agent called Instructo-Soar, that learns new tasks and other domain knowledge from natural language instructions. Instructo-Soar meets three key requirements of flexible instructability: it can (A) take any command at each instruction point, (B) handle instructions that apply to either the current situation or a hypothetical one (e.g., conditionals), and (C) learn each type of knowledge it uses (derived from its underlying computational model) from instructions.

Introduction

Tutorial instruction is a highly interactive dialogue that focuses on the specific task(s) being performed by a student. While working on tasks, the student may receive instruction as needed to complete tasks or to understand aspects of the domain or of previous instructions. This situated, interactive form of instruction produces very strong human learning [Bloom, 1984]. Thus, although it has received scant attention in AI, tutorial instruction has the potential to be a powerful knowledge source for intelligent agents.

Much of tutorial instruction's power comes from its *communicative flexibility*: the instructor is free to communicate whatever kind of knowledge a student needs at whatever point it is needed. The challenge is designing a tutorable agent that supports the wide breadth of interaction and learning abilities required by this flexible communication of knowledge. Our ultimate goal is to produce fully flexible tutorable agents, that can be instructed in the same ways as human students. A complete description of the properties and associated requirements of tutorial instruction is given by Huffman [1994]. In this paper, we focus specifically on three crucial requirements of flexible instructability:

A. **Command flexibility.** The instructor should be free to give any appropriate commands to teach a task. Commands should not be limited to only directly performable/observable actions (as in [Redmond, 1992; Mitchell *et al.*, 1990; Segre, 1987]), but may request unknown procedures or actions that cannot be performed until after other actions.

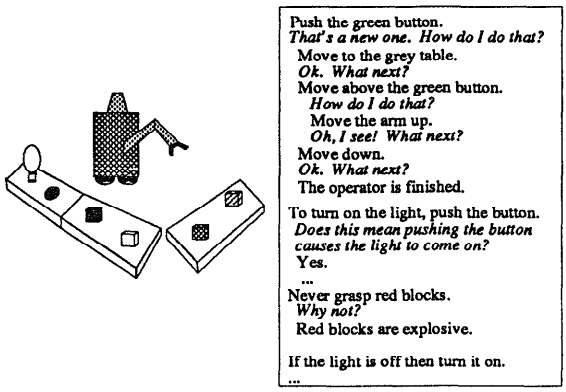
B. **Situation flexibility.** The instructor should be free to use instructions that apply to either the current task situation, or some hypothetical situation specified by the instruction.[1] *Implicitly situated* instructions apply to the current situation (e.g., commands: "Close the door") [Huffman, 1994]. *Explicitly situated* instructions explicitly indicate aspects of a hypothetical situation they apply to (e.g., conditionals: "If the power is low, turn off the machine"). These are especially useful for teaching contingencies that have not come up during ongoing task performance (perhaps rare or dangerous ones).

C. **Knowledge-type flexibility.** The instructor should be free to communicate any type of knowledge through instruction. Thus, the agent must be able to *learn* any type of knowledge it uses to perform tasks (control knowledge, knowledge about objects/properties, knowledge of actions' effects, etc.) from instruction.[2]

*This work was sponsored by NASA/ONR contract NCC 2-517, and by a University of Michigan Predoctoral Fellowship.

[1] Human instructors use both of these options. In one protocol, for instance, 119 out of 508 instructions (23%) involved hypothetical situations, with the remainder applying to the current situation when given [Huffman, 1994].

[2] Because tutorial instructions provide knowledge for



Push the green button.
That's a new one. How do I do that?
Move to the grey table.
Ok. What next?
Move above the green button.
How do I do that?
Move the arm up.
Oh, I see! What next?
Move down.
Ok. What next?
The operator is finished.

To turn on the light, push the button.
Does this mean pushing the button causes the light to come on?
Yes.
...
Never grasp red blocks.
Why not?
Red blocks are explosive.

If the light is off then turn it on.
...

Figure 1: An example of tutorial instruction.

An agent that learns from tutorial instruction is a type of learning apprentice system (LAS) [Mitchell *et al.*, 1990]. LAS's learn by interacting with an expert; either observing the expert solving problems [Redmond, 1992; Mitchell *et al.*, 1990; Wilkins, 1990; Segre, 1987; VanLehn, 1987], or attempting to solve problems and allowing the expert to guide and critique decisions that are made [Laird *et al.*, 1990; Gruber, 1989; Kodratoff and Tecuci, 1987; Porter and Kibler, 1986]. Each LAS has learned particular types of knowledge: e.g., operator implementations [Mitchell *et al.*, 1990], goal decomposition rules [Kodratoff and Tecuci, 1987], operational versions of functional goals [Segre, 1987], control knowledge/features [Gruber, 1989], heuristic classification knowledge [Wilkins, 1990], etc.

Flexible tutorial instruction extends LAS's interaction and learning abilities in a number of ways. First, the instructor may specify unknown tasks or tasks with unachieved preconditions at any instruction point (requirement A). Past LAS's limit input to particular commands at particular times (e.g., only commanding directly executable actions) and typically do not allow unknown commands at all. Second, tutorial instruction allows explicitly situated instructions (requirement B); past LAS's do not. Third, past LAS's learn only a subset of the types of knowledge they use to perform tasks. Tutorial instruction's flexibility requires that all types of task knowledge can be learned (requirement C).

This paper describes how analytic and inductive learning techniques can be combined and embedded within an agent to produce general learning from flexible tutorial instruction. We present a learning approach called *situated explanation* (an extension of EBL) that utilizes the situation an instruction applies to and the larger instructional context to guide the learning process. The approach is implemented in an instructable agent called Instructo-Soar, built within Soar [Laird *et al.*, 1987]. Huffman and Laird [1993] described how Instructo-Soar supports (A) command flexibility, learning hierarchies of new tasks, and extending tasks to new situations, given imperative natural language commands like those at the top of Figure 1. This paper extends the theory underlying Instructo-Soar to cover (B) situation and (C) knowledge-type flexibility. The types of knowledge that the agent should be able to learn from instruction – namely, all the types it uses to perform tasks – are identified by examining the computational model underlying the agent. Instructo-Soar can in fact learn every type of knowledge it uses in task performance – control knowledge, operator knowledge, etc – from instruction. The claim is not that Instructo-Soar can learn any possible *piece* of knowledge of these types, but that it can learn examples of each type. By combining requirements A, B, and C, Instructo-Soar exhibits a breadth of capabilities needed for flexible instructability.

Expanding the requirements

Requirements A, B, and C imply a set of possibilities that a flexibly instructable agent must deal with. (A) Command flexibility allows the instructor to give any command at any point in the instructional dialogue.[3] For any command, there are three possibilities: (1) the commanded action is known, and the agent performs it; (2) the commanded action is known, but the agent does not know how to perform it in the current situation (the instructor has skipped steps that meet the commanded action's preconditions); or (3) the commanded action is unknown to the agent.

(B) Situation flexibility allows each instruction to apply to either the current situation or a hypothetical one. A *situation* consists of a starting state S and a goal G that the agent will move towards by performing the instructed step. Either S or G may be explicitly indicated, making for two types of explicitly situated instructions. For example, "If the light is on, push the button" indicates a hypothetical state with a light on; "To turn on the machine, flip the switch" indicates a hypothetical goal of turning on the machine.

(C) Knowledge-type flexibility allows instructions that can require the agent to learn any of the types of knowledge it uses. We have identified the knowledge types used in our agent by examining its underlying computational model, called the *problem space computational model* (PSCM) [Newell *et al.*, 1990; Yost and

particular situations, they are best for teaching tasks with *local control structure* (e.g., serial constructive tasks). Local decisions can combine to exhibit global control behavior [Yost and Newell, 1989], but teaching a global method as a sequence of local decisions is difficult. Thus, we focus on learning knowledge for tasks involving local control.

[3] Flexible *initiation* of each instruction – by either instructor or agent – is viewed as a separate requirement. The agent described here supports command flexibility for agent-initiated instruction only.

Entity	Knowl. type	Example
state	inference	gripper closed & directly above obj → holding obj.
operator	proposal	If goal is pick up obj on table1, and not docked at table1, then propose moving to table1.
operator	preferential	If goal is pick up small metal obj on table1, prefer moving to table1 over fetching magnet.
operator	effects	An effect of the operator move to table1 is that the robot becomes docked at table1.
operator	termination	Termination conditions of pick up obj: gripper raised & holding obj.

Table 1: The five types of knowledge of PSCM agents.

Newell, 1989]. The PSCM is a general formulation of the computation in a knowledge-level agent, and many applications have been built within it [Rosenbloom *et al.*, 1993]. Because its components approximate the knowledge-level, the PSCM is an apt choice for identifying an agent's knowledge types.[4] Soar is a symbol-level implementation of the PSCM.

A PSCM agent moves through a sequence of *states* in a *problem space*, by sequentially applying *operators* to the current state. Operators transform the state, and may affect the external world by producing motor commands. The agent reaches an *impasse* when its immediately available knowledge is not sufficient either to select or fully apply an operator. When this occurs, another problem space context – a *subgoal* – is created, with the goal of resolving the impasse. This second context may impasse as well, causing a third context to arise, and so on.

The only computational entities in the PSCM mediated by the agent's knowledge are states and operators. There are a small set of PSCM-level functions on these entities that the agent performs, each using a different type of knowledge. Thus, it is straightforward to enumerate the types of knowledge within a PSCM agent; they are listed in Table 1 ([Huffman, 1994; Yost and Newell, 1989] describe the derivation of this list). Because Soar is an implementation of the PSCM, the knowledge within Soar agents is of these types.

Thus, as listed in Table 2, our three requirements expand to three types of command flexibility, three types of situated instructions and five types of knowledge to be learned.

[4]This is not true of other computational models, such as Lisp or connectionist models.

(A) Command flexibility	known command	
	skipped steps	
	unknown command	
(B) Situation flexibility	implicitly situated	
	explicitly situated:	hyp. state
		hyp. goal
(C) Knowledge-type flexibility	state inference	
	operator proposal	
	operator preferential	
	operator effects	
	operator termination	

Table 2: Expanded requirements of instruction.

Learning from instructions

Learning from instruction involves both analytic learning (learning based on prior knowledge) and inductive learning (going beyond prior knowledge). Analytic learning is needed because the agent must learn general knowledge from specific instructions that combine known elements (e.g., learning to push buttons by combining known steps to push the green one). Inductive learning is needed because the agent must learn new task goals and domain knowledge beyond the scope of its prior knowledge. Our goal is not to produce more powerful analytic or inductive techniques, but rather to specify how these techniques come together to produce a variety of learning (requirement C) in the variety of instructional situations and interactions faced by a flexibly instructable agent (requirements A, B).

Analytic methods can produce high-quality learning from a single example (e.g., a single instructing of a procedure). Thus, we use an analytic approach – a type of explanation-based learning – as the core of our learning from instruction method, and fall back on inductive methods when explanation cannot be used. Because of requirement B, the explanation process is performed within the situation the instruction is meant to apply to – either the current task situation, or one explicitly specified in the instruction.

The resulting approach to learning from tutorial instruction, called *situated explanation*, is conceptually simple. For each instruction I, the agent first determines what situation I is meant to apply to, and then attempts to *explain* why the step indicated by I leads to goal achievement in that situation (or prohibits it, for negative instructions). If an explanation can be made, it produces general learning of some knowledge I_K by indicating the key features of the situation and instruction that cause success. If an explanation cannot be completed, it indicates that the agent lacks some knowledge, M_K, needed to explain the instruction. The lack of knowledge is dealt with in different ways depending on the instructional context, as described below.

In more detail: the situation (starting state S and a goal G to be reached) for each instruction I is produced by altering the current task situation to reflect

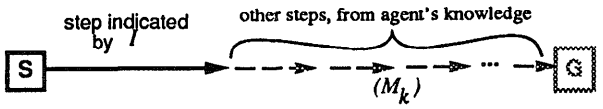


Figure 2: Situated explanation of instruction I in situation $[S, G]$.

any situation features specified in the instruction (e.g., state features like "If the light is off..."). Then, the agent attempts to explain I's use in $[S, G]$ using its prior knowledge. A PSCM agent's knowledge applies to the current situation to select and apply operators and to make inferences. When explaining an instruction I, this knowledge is applied internally to the situation $[S, G]$. That is, explanation takes the form of *forward internal projection*. As depicted in Figure 2, the agent "imagines" itself in state S, and then runs forward, applying the instructed step, and knowledge that it has about subsequent states/operators. If G is reached within this projection, then the projected path from S, through the step instructed by I, to G comprises an explanation of I. By indicating the features of I, S, and G causally required for success, the explanation allows the agent to learn general knowledge from I (as in standard EBL, realized in our agent by Soar's chunking mechanism [Rosenbloom and Laird, 1986]). For instance, if the agent projects "Move to the grey table" (and then the actions following it) to reach the goal of pushing a button, it learns a general operator proposal rule: "Propose `move-to-table(?t)` when trying to push button `?b`, `located-on(?b,?t)`, and `not docked-at(?t)`." This rule generalizes the instruction by removing the table's color (which was not causally important), and specializes it by including the fact that the button is on the table.

This method depends on the agent having knowledge to form an explanation. However, an instructable agent's knowledge will often be incomplete (otherwise, it would not need instruction). That is, it will often be missing one or more pieces of prior knowledge M_K (of any PSCM type) needed to complete the explanation of I. Missing knowledge (in Figure 2, missing arrows) will cause an incomplete explanation by precluding achievement of G in the projection. For instance, the agent may not know a key effect of an operator, or a crucial state inference, needed to reach G. More radically, the action commanded by I may be completely unknown and thus inexplainable.

There are a number of general options a learning system may follow when it cannot complete an explanation. (O1) It could delay the explanation until later, in hopes that the missing knowledge (M_K) will be learned in the meantime. Or, (O2-O3) it could try to complete the explanation now, by learning the missing knowledge somehow. The missing knowledge could be learned (O2) inductively (e.g. by inducing

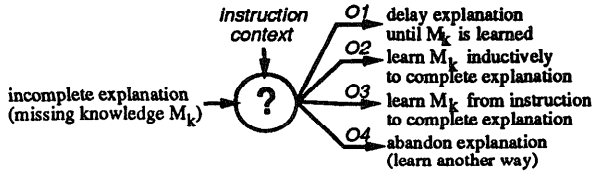


Figure 3: Options when faced with an incomplete explanation because of missing knowledge M_K.

over the "gap" in the explanation, as in [VanLehn *et al.*, 1992] and many others), or, (O3) in an instructable agent's case, through further instruction. Finally, (O4) it could abandon the explanation altogether, and try to induce the desired knowledge instead.

Given only an incomplete explanation, it would be difficult to choose which option to follow. Identifying the missing knowledge M_K in the general case is a difficult credit assignment problem, and there is nothing in the incomplete explanation that predicts whether M_K will be learned later if the explanation is delayed. However, as indicated in Figure 3, an instructable agent has additional information available to it besides the incomplete explanation itself. Namely, the instructional context (that is, the type of instruction and its place within the dialogue) often constrains M_K in a way that determines which of the four options (O1-O4) is most appropriate for a given incomplete explanation. For instance, the context can indicate that M_K is likely to be learned later, making delaying explanation the best option; alternatively, it can indicate that there are potentially many pieces of knowledge missing, making abandoning explanation a more reasonable option.

To describe how different instructional contexts can constrain dealing with incomplete explanations, the next section presents a series of four examples (one for each option) handled by the Instructo-Soar agent we have built. In describing different instructional contexts, the examples also illustrate Instructo-Soar's coverage of each piece of requirements A, B, and C.

Instructo-Soar: An instructable agent

Instructo-Soar is an instructable agent built within Soar – and thus, the PSCM – that uses the situated explanation approach outlined above. It is currently applied to a simulated robotic domain, with a Hero robot, blocks, tables, buttons, a light, and an electromagnet. Instructo-Soar begins with the knowledge of a set of primitive operators, such as moving its arm up and down, and opening and closing its hand. However, its knowledge of the effects of these operators on objects in the domain is incomplete. Also, it begins with no knowledge of complex operators, such as picking up or arranging objects.

Ex. 1. Delaying explanation while learning procedures. Instructo-Soar learns new procedures from

commands like those at the top of Figure 1, for "Push the green button." Huffman and Laird [1993] describe this learning in detail. When the command to push the button cannot be completed (because this procedure is unknown), the agent reaches an impasse and asks for further instruction. Instructions to perform the procedure are received and executed in turn (e.g., "Move to the grey table"). However, they cannot be explained: the agent does not know the goal of the procedure (e.g., the goal of "push the green button") – the endpoint of the explanation – or the steps following the current one to reach that endpoint.

In this instructional context – explaining a commanded step of a procedure being learned –it is clear that the missing knowledge of the following steps and the procedure's goal *will* be acquired later, because the instructor is expected to teach the procedure to completion. Thus, the agent *delays explanation* (option O1) and for now memorizes each instruction in a rote, episodic form, to be recalled and explained later. At the end of the first execution of a procedure, the instructor indicates "The operator is finished", and the agent uses a simple heuristic (comparing the initial and final states) to induce the goal concept (termination conditions) of the new procedure (e.g., the gripper is directly above the button). Since this induction may be incorrect, it is presented to the instructor, who may add or remove features before verifying the result (not shown in Figure 1).

Now that the goal and all steps are known, the original instructions are recalled and explained, via forward projection from the initial state. If this projection succeeds, the agent learns general implementation rules for the procedure (a series of proposals for sub-operators: `move-to-table`, `move-arm(above)`, ...) based on the features in the projection (a-la EBL).[5]

The parts of (A) command flexibility from Table 2 are supported through recursive instruction/explanation. Any incompletable command causes an impasse and leads to further instruction/explanation within a subgoal. There may be multiple levels of embedded impasses, supporting hierarchical instruction. For example, within the instructions for "push button", the command "Move above the green button" cannot be completed because of a skipped step (the arm must be raised to move above something). An impasse arises where the instructor indicates the needed step, and then continues instructing "push button". Since new or previously learned operators can be commanded while teaching a larger operator, operator hierarchies can be taught. Instructo-Soar has learned hierarchies involving lining up, picking up, grasping, and putting down objects, etc.

[5]If the projection fails, there is additional missing knowledge (e.g., unknown operator effects). Since it is extremely difficult to localize the missing knowledge, the agent abandons explanation and uses simple heuristics (described below) to induce proposals for the sub-operators.

The parts of requirements B and C in Table 2 met by learning a new operator are implicitly situated instructions, and learning the operator's termination conditions, and proposals for its sub-operators. The remaining examples illustrate Instructo-Soar's handling of explicitly situated instructions, and learning the other types of PSCM knowledge.

Ex. 2. Completing explanations by inducing missing knowledge. Instructo-Soar's domain includes a light that is toggled using a button. The agent has been taught how to push the button, but does not know its effect on the light. Now it is told: `To turn on the light, push the red button.` This instruction specifies a *hypothetical goal* of turning on the light. Based on the instruction, the agent creates a situation consisting of that goal and a state like the current state, but with the light off.[6] From this situation, the agent forward projects the action of pushing the button. However, since the agent is missing the knowledge (M_K) that pushing the button affects the light, the light does not come on within the projection. Thus, the explanation is incomplete.

In this case, the form of the instruction – a purpose clause specifying a hypothetical goal – allows the agent to form a strong hypothesis about the missing knowledge. Based on the instruction, the action of pushing the button is expected to *generate* turning on the light.[7] That is, the agent expects the specified goal (light on) to be met after performing the single push-button action. This expectation constrains the "gap" in the explanation: the state after pushing the button *should* be a state with the light on, and only one action was performed to produce this effect. Because the effect is not achieved, the most straightforward inference of M_K is that the single action – pushing the button – *causes* any unachieved goal features to be met (e.g., it causes the light to come on). The instructor is asked to verify this inference.[8] If verified, the agent learns new piece of operator effect knowledge:

```
if   projecting push-button(?b), color(?b,red),
     and the light is not on
then the light is now on.
```

Immediately after being learned, this rule applies to the light in the forward projection for the current instruction. The light comes on, completing the instruction's explanation by achieving its goal. From this explanation, the agent learns to propose pushing the red button when its goal is to turn on the light.

This example illustrates Instructo-Soar's coverage of hypothetical goal instructions for requirement B, and learning operator effects for requirement C.

[6]Hypothetical situations are created from language in a straightforward way; for details, see Huffman [1994].

[7]DiEugenio [1993] found empirically that this expectation holds for 95% of naturally occurring purpose clauses.

[8]If the inference is rejected, the agent assumes there must be extra steps involved to turn on the light, and learns to propose pushing the button as the first step.

Ex. 3. Abandoning explanation. The agent abandons an incomplete explanation in two cases: (1) when the incompleteness could be caused by missing knowledge about any of multiple operators; and (2) when the instructor declines to give further instruction about the missing knowledge upon being asked. An example of (1) was mentioned earlier: after learning the steps and the goal of a new procedure, sometimes the agent still cannot explain the steps. This explanation failure could be caused by missing knowledge about *any* of the steps, making it difficult to isolate. Thus, it is reasonable to abandon explaining the steps, and instead to induce a proposal rule for each step. Simple heuristics are used to select features of the state, goal, and step. For instance, one heuristic selects the relationship between arguments of the goal (e.g., the button in "Push the green button") and the step (e.g., the table in "Move to the grey table") to include as a condition for proposing the step (e.g., including `located-on(?b,?t)`).

As an example of (2) the instructor declining to give further instruction, consider these instructions:

`Never grasp red blocks.`
Why? `Trust me.`

"Never grasp" prohibits a step from applying to a hypothetical situation in which it might apply (as do negative imperatives in general). Thus, Instructo-Soar creates a *hypothetical state* with a red block that can be grasped. (Instructo-Soar also handles positive hypothetical state instructions; e.g., "If the light is off, ...") Since no goal is specified by the instruction, and there is no current goal, a default goal of "maintaining happiness" is used. From this hypothetical situation, the agent internally projects the "grasp" action. The resulting state, in which the agent is grasping a red block, is acceptable ("happy") according to the agent's knowledge. Thus, the projection does not explain why the action is prohibited. The agent asks for further instruction, in an attempt to learn M_K and complete the explanation.

The instructor can decline to give further information by saying `Trust me`. Since the instructor will not provide M_K, the agent is forced to abandon the explanation. Instead, it induces conditions for prohibiting "grasp". Instructo-Soar conservatively guesses that every feature of the hypothetical state is relevant to rejecting the instructed operator. Thus, it induces that it should reject grasping `blocks` with `color red`.

Ex. 4. Completing explanations through further instruction. Alternatively, the instructor could provide further instruction, saying for instance `Red blocks are explosive`. From this instruction, the agent learns a state inference rule: `blocks` with `color red` have `explosiveness high`. Instructo-Soar learns inferences from simple statements, and from conditionals like "If the magnet is powered and directly above a metal block, then the magnet is stuck to the block," by essentially translating the utterance directly into a rule.[9] State inference instructions may be used to introduce new features (e.g, `stuck-to`) that extend the agent's representation vocabulary.

The rule learned from "Red blocks are explosive" adds `explosiveness high` to the block that the agent had simulated grasping in the hypothetical situation. The agent knows that touching an explosive object may cause an explosion – a negative result. This negative result completes the explanation of "never grasp," and the agent learns to avoid grasping objects with `explosiveness high`.

Completing the explanation through further instruction (this example), produces more general learning than in example 3. In this example, if the agent is later told `Green blocks are explosive`, it will avoid grasping them as well. In general, multiple levels of instruction can lead to higher quality learning than a single level, because learning is based on an explanation composed from lower level knowledge (M_K) rather than inductive heuristics alone.

Examples 3 and 4 illustrate hypothetical state instructions (requirement B), and learning operator preferential knowledge (here, operator rejection) and state inferences (requirement C).

Discussion

These examples have shown how an instruction's situation and context can affect the process of learning from it. First, the *situation* an instruction applies to provides the endpoints for attempting to explain the instruction.

Second, different instructional contexts can indicate which option to follow when the explanation of an instruction cannot be completed. For instance, the context of learning a new procedure indicates that delaying explanation is best, since the full procedure will eventually be taught. After learning the full procedure, if a step cannot be explained, missing knowledge could be anywhere in the procedure, so it is best to abandon explanation and learn another way. Purpose clause (hypothetical goal) instructions localize missing knowledge, by giving strong expectations about a single operator that should achieve a single goal; this makes it plausible to induce missing knowledge and complete the explanation. In cases other than these, the default is to ask for instruction about missing knowledge to complete the explanation. Even then, if the instructor declines, the explanation must be abandoned.

The examples also illustrate coverage of Table 2, within one domain. Although the domain is simple, it does give rise to a range of the different types of instructional interactions and learning that occurs in tutorial instruction. We claimed that tutorial instruc-

[9] This occurs by chunking, but in an uninteresting way. Instructo-Soar does not use explanation to learn state inferences. An extension would be to try to explain why an inference holds using a deeper causal theory.

tion's flexibility requires a breadth of learning and interaction capabilities in an instructable agent. Empirically, combining (A) command flexibility, (B) situation flexibility, and (C) knowledge-type flexibility, Instructo-Soar displays *nineteen* distinct instructional capabilities [Huffman, 1994]. These are primarily combinations of the requirements in Table 2. For instance, Instructo-Soar can learn three different kinds of knowledge from hypothetical state instructions (inferences, proposals, and rejections). (In addition to these combinations there are supporting behaviors, like taking instructions to alter/verify inferences.) This myriad of instructional behavior does not require nineteen different learning techniques, but arises from applying a single technique, situated explanation in a PSCM agent, across a range of instructional situations.

As an evaluation criterion for instructable agents, we have developed a set of requirements for a "complete tutorable agent" [Huffman, 1994]. This paper focuses on three key requirements; there are eleven in all. Instructo-Soar meets seven of the eleven either fully or partially.

The unmet requirements provide impetus for further work. First, a complete tutorable agent must deal with instructions in all their linguistic complexity. We have applied our techniques to a small domain, with simple actions and language; in more complex domains, more complex language will be needed. Second, a complete tutorable agent must allow instructions to be initiated by either the agent or the instructor. Currently, Instructo-Soar's dialogue is driven completely by the agent, which receives instruction only when it notices a lack of knowledge. Third, a complete tutorable agent must deal with both incomplete and incorrect knowledge. Instructo-Soar's learning thus far incrementally adds knowledge to an incomplete theory. We have just begun work on correcting incorrect domain knowledge through instruction.

Acknowledgments: Thanks to Jay Runkel and Randy Jones for helpful comments on earlier drafts.

References

[Bloom, 1984] B. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Ed. Researcher*, 13(6):4–16, 1984.

[DiEugenio, 1993] B. DiEugenio. *Understanding NL Instructions: A computational approach to purpose clauses.* PhD thesis, Univ. of Pennsylvania, 1993.

[Gruber, 1989] T. Gruber. Automated knowledge acquisition for strategic knowledge. *Machine Learning*, 4(3-4):293–336, 1989.

[Huffman and Laird, 1993] S. Huffman and J. Laird. Learning procedures from interactive natural language instructions. In *Machine Learning: Proc's. 10th Intl. Conference*, 1993.

[Huffman, 1994] S. Huffman. *Instructable Autonomous Agents.* PhD thesis, Univ. of Michigan, January 1994.

[Kodratoff and Tecuci, 1987] Y. Kodratoff and G. Tecuci. Techniques of design and DISCIPLE learning apprentice. *Intl. J. Expert Systems*, 1(1):39–66, 1987.

[Laird *et al.*, 1987] J. Laird, A. Newell, and P. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Laird *et al.*, 1990] J. Laird, M. Hucka, E. Yager, and C. Tuck. Correcting and extending domain knowledge using outside guidance. In *Proc. 7th Intl. Conf. on Machine Learning*, 1990.

[Mitchell *et al.*, 1990] T. Mitchell, S. Mahadevan, and L. Steinberg. LEAP: A learning apprentice system for VLSI design. In Y. Kodratoff and R. Michalski, editors, *Machine learning: An AI approach, Vol. III.* Morgan Kaufmann, 1990.

[Newell *et al.*, 1990] A. Newell, G. Yost, J. Laird, P. Rosenbloom, and E. Altmann. Formulating the problem space computational model. In *Proc. 25th Anniversary Symposium, CMU School of Computer Science*, 1990.

[Porter and Kibler, 1986] B. Porter and D. Kibler. Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning*, 1:249–286, 1986.

[Redmond, 1992] M. Redmond. *Learning by observing and understanding expert problem solving.* PhD thesis, Georgia Tech, 1992.

[Rosenbloom and Laird, 1986] P. Rosenbloom and J. Laird. Mapping EBG onto Soar. In *AAAI-86*, 1986.

[Rosenbloom *et al.*, 1993] P. Rosenbloom, J. Laird, and A. Newell, editors. *The Soar Papers: Research on integrated intelligence.* MIT Press, 1993.

[Segre, 1987] A. Segre. A learning apprentice system for mechanical assembly. In *IEEE Conf. on AI for Applications*, pages 112–117, 1987.

[VanLehn *et al.*, 1992] K. VanLehn, R. Jones, and M. Chi. A model of the self-explanation effect. *J. of the Learning Sciences*, 2(1):1–59, 1992.

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

[Wilkins, 1990] D. Wilkins. Knowledge base refinement as improving an incomplete and incorrect domain theory. In Y. Kodratoff and R. Michalski, editors, *Machine learning: An AI approach, Volume III.* Morgan Kaufmann, 1990.

[Yost and Newell, 1989] G. Yost and A. Newell. A problem space approach to expert system specification. In *IJCAI-89*, 1989.