# Instructable Autonomous Agents

by

Scott Bradley Huffman

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
January 1994

Doctoral Committee:

Associate Professor John E. Laird, Chair
Assistant Professor W. P. Birmingham
Professor R. K. Lindsay
Professor W. Rounds
Associate Professor E. Soloway

## Abstract

INSTRUCTABLE AUTONOMOUS AGENTS

by
Scott Bradley Huffman

Chair: John E. Laird

In contrast to current intelligent systems, which must be laboriously programmed for each task they are meant to perform, instructable agents can be *taught* new tasks and associated knowledge. This thesis presents a general theory of learning from tutorial instruction and its use to produce an instructable agent. Tutorial instruction is a particularly powerful form of instruction, because it allows the instructor to communicate whatever kind of knowledge a student needs at whatever point it is needed. To exploit this broad flexibility, however, a tutorable agent must support a full range of interaction with its instructor to learn a full range of knowledge. Thus, unlike most machine learning tasks, which target deep learning of a single kind of knowledge from a single kind of input, tutorability requires a *breadth* of learning from a broad range of instructional interactions.

The theory of learning from tutorial instruction presented here has two parts. First, a computational model of an intelligent agent, the *problem space computational model*, indicates the types of knowledge that determine an agent's performance, and thus, that should be acquirable via instruction. Second, a learning technique, called *situated explanation*, specifies how the agent learns general knowledge from instruction. The theory is embodied by an implemented agent, Instructo-Soar, built within the Soar architecture. Instructo-Soar is able to learn hierarchies of completely new tasks, to extend task knowledge to apply in new situations, and in fact to acquire every type of knowledge it uses during task performance – control knowledge, knowledge of operators' effects, state inferences, etc. – from interactive natural language instructions. This variety of learning occurs by applying the situated explanation technique to a variety of instructional interactions involving a variety of types of instructions (commands, statements, conditionals, etc.). By taking seriously the requirements of flexible tutorial instruction, Instructo-Soar demonstrates a breadth of interaction and learning capabilities that goes beyond previous instructable systems, such as learning apprentice systems. Instructo-Soar's techniques could form the basis for future "instructable technologies" that come equipped with basic capabilities, and can be taught by novice users to perform any number of desired tasks.

# Acknowledgements

My first thanks must go to my advisor, John Laird. John has taught me how to do research: how to analyze a problem, understand its dimensions, build systems to attack it, and learn from those systems. John's unshakable optimism helped me to press on at many points when I was ready to throw in the towel. His upbeat demeanor is a large part of what makes Michigan's AI Lab such a great place to work. Finally, as important as anything else, John has taught me that it is possible to be a successful scientist and still have a life outside the lab.

I've been blessed to have an excellent group of people to work with, both at the AI Lab and in the Soar group around the world. I have never lacked smart people to talk about problems with or fun people to take a break with. In particular, Rick Lewis, Craig Miller, and Doug Pearson are great friends that I've had very fruitful research interactions with.

For financial support, I owe thanks to the National Science Foundation for a graduate fellowship, to the Rackham graduate school at the University of Michigan for two fellowships, and to NASA Ames.

Over the years, my family has been a constant and continuous source of encouragement and support. I'm very thankful for them. I am also thankful for my church family at Cornerstone Christian Church. In addition to being a great group of friends, they have helped me to keep my focus on what's really important in life while working on this degree.

Finally, my greatest thanks goes to God, who gave me the strength and ability to do this work.

# Table of Contents

# List of Appendices

**Appendix**

# Chapter 1

# Introduction

The intelligent, autonomous agents of the future will be called upon to perform a wide and varying range of tasks, under a wide range of circumstances, over the course of their lifetimes. Performing these tasks requires knowledge. If the number of possible tasks and circumstances is large and variable over time (as it will be for a general, flexible agent), it becomes nearly impossible to pre-program an agent with all of the required knowledge. Thus, the agent needs the ability to learn as a part of its ongoing performance.

One powerful way that people acquire the knowledge needed to perform new tasks is through *tutorial instruction*. Tutorial instruction is a highly interactive dialogue that focuses on the specific task(s) being performed. Its power comes from allowing the instructor to communicate whatever kind of knowledge a student needs at whatever point it is needed. The challenge for a tutorable agent is to support the full breadth of interaction and learning that arises from this flexible communication of knowledge.

The goal of this thesis is to present a general theory of learning from tutorial instruction. The theory has been developed by determining both general and specific requirements that tutorability places on intelligent agents. These requirements encompass the ways an agent interacts with its instructor, comprehends instructions, and learns from them. The requirements serve as evaluation criteria for theories of instructional learning and implemented instructable systems, including the one described here.

Thus, the work to be presented has four major parts:

1. A specification of the problem of learning from tutorial instruction, and of the requirements it places on a complete tutorable agent.

2. A theory of learning from tutorial instruction. The theory includes a computational model of an intelligent agent and a learning technique for learning from instruction. The computational model indicates the types of knowledge that determine an agent's performance, and thus, that should be acquirable via instruction. The learning technique specifies how the agent acquires general knowledge from instructions.

3. An implemented instructable agent that embodies the theory. The agent, called Instructo-Soar [Huffman and Laird, 1993b; Huffman and Laird, 1993a], is built within the Soar architecture [Laird *et al.*, 1987]. Instructo-Soar learns to perform new tasks, extends known tasks to apply in new situations, and acquires a variety of types of domain knowledge, from interactive natural language instructions.

4. An evaluation of the agent based on the requirements for complete tutorability. The evaluation indicates that Instructo-Soar supports a broader range of instructional interaction and learning than previous instructable systems. In particular:

    a. The agent can learn each of the types of knowledge it uses in task performance from instruction.

b. The agent supports fully flexible interaction for instructions commanding actions, allowing the instructor to give any command at any point during instruction, whether or not the agent knows the how to perform the commanded task.

c. The agent can learn both from instructions that apply to the situation at hand (e.g., imperative commands like "Do X now"), and from instructions that apply to specified hypothetical situations (e.g., conditionals like "If the situation is ..., then do X").

These capabilities combine to produce *nineteen* distinct interaction and learning behaviors. A key contribution is that this breadth of behavior is produced not by nineteen different mechanisms, but by a single theory applied across a breadth of tutorial learning tasks.

A secondary goal has been to evaluate the theory as a cognitive model of human tutorial learning [Huffman *et al.*, 1993a]. The results are broadly consistent with what is known about human procedural learning, and make some specific behavioral predictions. These are discussed in Chapter 5.

The remainder of this chapter discusses the properties and requirements of tutorial instruction. An analysis of the task faced by a tutorable agent identifies three major task components, corresponding to capabilities for language comprehension, interaction, and learning. For each of these components, the requirements for complete tutorability are identified. The chapter concludes with a discussion of related work in AI and cognitive science, and an outline of the rest of the thesis.

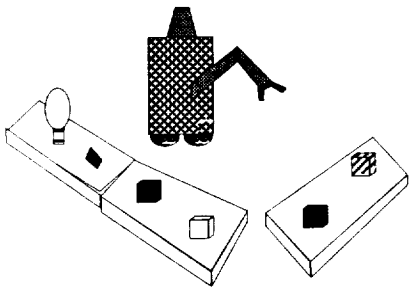
## 1.1 Properties of tutorial instruction

For the purposes of this thesis, tutorial instruction is defined as instruction given to an agent as it attempts to perform tasks in some domain. The instruction is both *situated* and *interactive*. It is *situated* in that it applies to particular task situations that arise while the agent is acting in the domain. It is *interactive* in that the agent may request instruction as needed to complete tasks or to understand aspects of the domain or of previous instructions. This type of instruction is common in dialogues between experts and apprentices working on various tasks [Grosz, 1980; Grosz, 1977]. A segment of tutorial instruction, given to the Instructo-Soar agent within a robotic domain, is shown in Figure 1.1.

Tutorial instruction can be viewed as a way of incrementally eliciting knowledge from an instructor. It has a number of properties that make this elicitation easy and flexible for the instructor:

P1. **Situation specificity.** Instructions are given for particular tasks in particular situations. To teach a task, the instructor need only provide suggestions for the specific situation at hand, rather than producing a global procedure that includes general conditions for applicability of every step, handling of all possible contingencies, etc. A number of authors have described the advantages of situation specific knowledge elicitation (e.g., [Davis, 1979; Gruber, 1989]). When instructions apply to the current situation, the situation can help to disambiguate them during their interpretation. Actions are performed in the world when they are instructed, giving the instructor direct feedback about their effects. Thus, when teaching a task, the instructor does not need to predict the effects of long instruction sequences, or reason about the effects an action might have in other situations.

P2. **Situation specification as needed.** Although instructions typically apply to the situation at hand, the instructor has the freedom to specify other situations as needed. For instance, the instructor may wish to indicate a course of action for a contingency that has not come up during ongoing task performance (perhaps because it is rare or dangerous). Conditional instructions are an example: "If the fluid in the cutting machine runs low, turn it off immediately." This type of instruction has been termed *explicitly situated* [Huffman and Laird, 1992]; the language of the instruction explicitly indicates aspects of the situation it applies to. Explicitly situated instructions allow the instructor to specify different levels of generality, from general advice, such as "Never touch a hot stove," or "Whenever the light is on, push the button," to specific steps within performance of specific tasks, such as "After you have assembled its housing, place the gasket over the left manifold."



Pick up the red block.
*How do I do that?*
Move to the yellow table.
*Ok. What next?*
Move the arm above the red block.
*How do I do that?*
Move up.
*Oh, I see! What next?*
Move down.
*Ok. What next?*
Close the hand.
*Ok. What next?*
Move up.
*Ok. What next?*
The operator is finished.

Go home.
Push the button.
*How do I do that?*
Move to the grey table.
*Ok. What next?*
Move above the button.
*Ok. What next?*
Move down.
*Ok. What next?*
The operator is finished.

To turn on the light, push the button.
*Does this mean that pushing the button causes the light to come on?*
Yes.

If the magnet is on, turn it off.
...

Figure 1.1: An example of tutorial instruction.

P3. **Incremental as-needed elicitation.** Knowledge acquisition occurs incrementally as a part of the agent's ongoing performance. Instructions are given when the agent is unable to perform a task; thus, they directly address points where the agent's knowledge is lacking.

P4. **Task structuring by instructor.** The instructor is free to structure larger tasks into smaller subtasks in any way desired. For instance, a task requiring ten primitive steps may be taught as a simple sequence of the ten steps, or as two subtasks of five steps each, etc. Since the agent can request additional instruction as needed, the instructor can indicate whatever action/subtask is deemed appropriate in a given situation, without worrying whether the agent already has knowledge of the action/subtask. If the agent does not know what an instructed action is, or how to perform it in the situation at hand, more instruction will be requested.

P5. **Knowledge-level interaction.** The instructor provides knowledge to the agent at the knowledge level [Newell, 1981], because the instructions refer to objects and actions in the world, not to symbol-level structures (e.g., data structures) within the agent. The interaction occurs in natural language, the language that the instructor uses to talk about the task, rather than requiring artificial terminology and syntax to specify the agent's internal data structures and processes.

Because tutorial instruction typically provides knowledge applicable to the agent's current situation or a closely related one, it is most appropriate for tasks with a *local control structure*. In tasks with local control structure, control decisions are made based on presently available information. Global control strategies, on the other hand, either follow a fixed regime or require an aggregation of information from multiple problem solving states to make control decisions. Local control structure is characteristic of constructive synthesis tasks, in which primitive steps are composed one after another to form a complete solution. In contrast, problem solving methods like constraint satisfaction and heuristic classification involve global control.

It is possible to produce a global control strategy using a combination of local decisions [Yost and Newell, 1989]. However, *teaching* a global method by casting it purely as a sequence of local decisions may be difficult. To acquire knowledge for tasks that involve global control, it may be more efficient to use a method-based knowledge acquisition tool (e.g., [Eshelman *et al.*, 1987; Musen, 1989; Marcus and McDermott, 1989; Birmingham and Siewiorek, 1989]) with the control strategy built in. This thesis focuses on instructing tasks with local control structure.

## 1.2 An analysis of the instructable agent's task

While allowing the instructor to provide knowledge in an easy and flexible way, the properties of tutorial instruction described above place severe requirements on an instructable agent. Such an agent is required to interact with its instructor, to behave or respond appropriately to instructions it is given, and to learn general knowledge from these instructions for use in future situations. Thus, the agent must perform three conceptually distinct but interrelated functions, each arising from various tutorial instruction properties described above:

1. The operations, objects, and relations specified in each instruction must be mapped onto the corresponding entities in the agent's internal representation of the task and context, as illustrated in Figure 1.2. This is required for the agent to apply the knowledge that is communicated by the instruction at the knowledge level (property P5) to its internal processing. The mapping task can be difficult, involving all of the problems of linguistic interpretation. In general, it may involve reasoning about the task and domain itself. For example, the agent may need to use task knowledge to find an operator with specified effects, or pragmatic knowledge to resolve referents.

   This is the **mapping problem**.

2. The agent must interact with the instructor in a flexible way to obtain instructions and advice. This interaction may be driven by the agent or by the instructor, and occurs during the

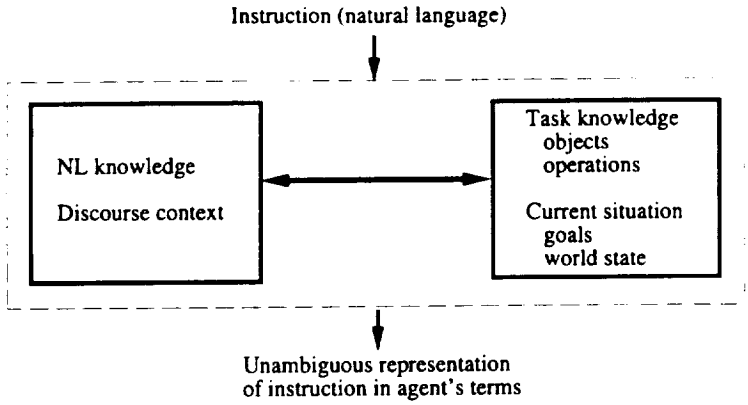



Figure 1.2: The mapping problem.

agent's ongoing performance to address its lacks of knowledge (property P3). The agent must be flexible in handling instructions that relate to the ongoing task situation and instructional dialogue in different ways; for instance, that apply to different kinds of situations (properties P1 and P2) or that structure tasks in different ways (property P4).

This is the **interaction problem**.

3. Finally, the agent should learn as much general long-term knowledge as possible from instructions. Many types of knowledge may be learned, since instructions can provide any type of knowledge that the agent is lacking (property P3). From instructions that apply to specific situations (property P1), the agent must learn general knowledge that will transfer to appropriate situations that arise in the future. Thus, the scope of applicability of each instruction must be accurately determined. To what range of situations does it apply? That is, which situational features are important for its application, and which can be generalized over?

   This is the **transfer problem**.

These problems are distinct but closely interrelated. For instance, if instructions that carry a certain type of knowledge cannot be mapped, then they cannot be given (limiting interaction) or learned from (limiting transfer). Limiting interaction to certain types of instructions at certain times may make it difficult to learn some type of task knowledge (limiting transfer), because the instructor cannot teach it in a natural way.

In what follows, requirements for a complete solution to each of these three problems are identified. Taken together, these requirements form the criteria for a *complete tutorable agent*. The following sections also indicate which requirements are met by the theory described in the rest of the thesis. This research has focused on the transfer problem and key parts of the interaction problem; approaches to the mapping problem and to the most open-ended aspects of the interaction problem are less developed.

### 1.2.1 The mapping problem

The mapping problem is the problem of transforming language input into the internal language of the agent. Both the domain and the range of the mapping make the problem difficult. First, natural language is complex, making its interpretation difficult. Second, the information that instructions are mapped to may be complex, containing a variety of parameters, constraints, and conditions.

Solving this problem in general is extremely difficult, and has not been the focus of this work. Rather, fairly standard natural language processing (NLP) techniques have been applied to allow communication about a sufficient range of actions and situations to demonstrate the learning theory.

The mapping problem will only be discussed as needed in the rest of the thesis to explain the implemented agent's operation. However, as it is crucial for the construction of tutorable agents, some of the general issues involved are discussed here. A number of sample instructions are used as examples. These have been collected from various sources: protocols of subjects instructing a simulated robot; a teacher instructing a student to use a flight simulator; a cookbook, a computer manual, and various device manuals.

### Complexity of natural language

Because of language's flexibility, instructions may be expressed in many different ways. Just and Carpenter [1976] point out that complicated instructions can be given for very simple tasks, making "following the instructions" primarily a comprehension task. In general, instruction may contain difficult constructions such as indirect speech acts, analogies and metaphors, etc.

Even if these types of constructions are disallowed, two difficult problems in interactive instruction are referent resolution and incompleteness. Referent resolution - determining what a noun phrase refers to - has been the topic of multiple Ph.D. theses (e.g., [Gershman, 1979; Grosz, 1977]). Grosz [1980,1977] specifically targeted referent resolution in expert/apprentice dialogues. Many standard reference problems appear in the sample instructions that have been collected, such as anaphors, comparatives and superlatives, generics, references to composite constructions, and episodic references.

Incompleteness is the problem of an action or other information being incompletely specified by an instruction. This is extremely common; the instructor expects the student to fill in details that are left out, using linguistic context, knowledge of the domain, or perhaps domain-general common sense knowledge. For example:

- *Move arm to the right.* How far?
- *Sprinkle with seasoned salt.* Sprinkle what? How much?
- *Pull back left.* How fast? For how long?

Some of these are examples of ellipsis, where the missing information can be filled in by looking at the linguistic context [Frederking, 1988]. Other cases require domain knowledge or task context. For example, in *Sprinkle with seasoned salt,* the agent must use domain knowledge about seasoned salt, perhaps about what amount is typically used, to further specify the sprinkling action. Martin and Firby [1991] and Chapman [1990] describe techniques for using the current execution context to deal with incompleteness; DiEugenio and Webber [1992] discuss a process for interpreting some types of incomplete instructions using domain knowledge. However, no system has dealt with the full range of incomplete instructions.

### Complexity of information being mapped

In addition to the complexity of natural language, the mapping problem is difficult because the information carried by instructions is complex. For example, specifying an action in an instruction may require various arguments (objects, instruments, etc); actions may be repetitive, conditional, concurrent with other events; they may have duration, rate, directionality, and so on. Language is flexible enough to express any of these parameters.

It would be difficult (and it is outside of the scope of this work) to characterize all of the possible types of actions and their parameters, linguistically or functionally. However, a few examples from our instruction samples, classified in an ad-hoc manner, will illustrate the complexity and variability:

1. Actions with definitive start and end points and standard parameters (e.g., actor, object, instrument, location).
   - *Pick up the red block.*
   - *Melt butter in a skillet.*

| |
|---|
| 1. main action/state |
| 2. cause |
| 3. manner |
| 4. purpose |
| 5. result |
| 6. figure (for motion event) |
| 7. path (for motion event) |
| 8. ground (for motion event) |
| 9. degree of realization ("barely", "almost") |
| 10. phase (starting vs. stopping, initiating vs. finishing) |
| 11. aspect (the pattern of distribution through time of an action or state; "hit" vs. "beat") |
| 12. rate |
| 13. causativity (event occurs itself vs. volitionally) |
| 14. person - relation between an actor and the speaker. E.g. "I", "you", "they." |
| 15. relation to comparable events (whether action occurs alone, in addition to, or in place of comparable events - "He only danced", "He also danced", etc.). |
| 16. temporal setting |
| 17. spatial setting |
| 18. direction (of motion) |

Table 1.1: Some of Talmy's semantic categories for motion constructions [Talmy, 1985]

- *Turn back to the airport.*

2. Actions expressed as constraints to satisfy.
   - *Place it on the rightmost side of the red block, so it's still standing.*
   - *Try to keep the horizon just exactly perpendicular.*
   - *Load a cassette with the edge where the tape is exposed down and the side you want to record on facing you.*
3. Negative expressions that constrain action.
   - *Keep the unit as far as possible from a TV set.*
   - *You don't want to use flaps unless you're taking off.*
   - *Be careful to make your password something that is not obvious.*
4. Continuous actions with explicit monitoring conditions delineating their end point.
   - *Adjust the REC LEVEL controls so that each meter needle deflects no further than 0 VU.*
   - *Hit thrust until we're at full throttle.*
   - *Add coconut and saute until light brown, stirring constantly.*

Unfortunately, a complete characterization of actions that may need to be mapped from instruction is hard to find. There have been some limited attempts to characterize the space of possible actions in AI (e.g., [Schank, 1975; Wilks, 1975]) and linguistics (e.g, [Talmy, 1985; Miller and Johnson-Laird, 1976]). Talmy [1985], for instance, analyzes the semantics of utterances specifying motion, laying out a dizzying *thirty-seven* motion-related semantic categories. Some of the most relevant are shown in Table 1.1. There is clearly a great amount of work to be done on understanding the space of possible actions and their parameters, and how linguistic constructions map onto this space.

Defining the requirements that the mapping problem places on a completely tutorable agent is straightforward. A complete tutorable agent must have the ability to correctly comprehend and map all aspects of an utterance that fall within the scope of information it can possibly represent. An agent cannot be expected to interpret aspects that fall outside its representation abilities. For example, if an agent has no representation of aspect (number 11 in Table 1.1), it cannot be expected

to correctly map the difference between "Hit the drum" and "Beat the drum." Representation abilities may be augmented through instruction, but this occurs by building up from existing abilities.

This work has not emphasized the mapping problem, and Instructo-Soar does not meet the complete mapping requirement, but rather performs mapping as needed to demonstrate its other capabilities.

### 1.2.2 The interaction problem

The interaction problem is the problem of supporting flexible dialogue with an instructor. An instructable agent moves towards a complete solution to the interaction problem to the degree that it provides freedom of interaction to the instructor. The properties of flexible interaction can be specified in terms of individual instruction events, where an instruction event is the receiving of a single instruction at a particular point in the ongoing task and discourse context. A complete tutorable agent must be able to handle any instruction event that is coherent at the current point in the ongoing discourse. Each instruction event:

1. Is *initiated* by someone – either the student or the teacher.
2. *Carries* a piece of knowledge of a particular type; e.g., a command to do a particular action, an inference to be made in a particular state, etc. (knowledge types are described more fully in Chapter 2).
3. *Applies* to a situation – either the current situation at hand or some specified situation.

"Any possible" instruction event can be generated by taking the cross product of possibilities of each of these three dimensions. Thus, a complete tutorable agent must support instruction events with:

1. **Flexible initiation.** Instruction events can be initiated by agent or instructor.
2. **Flexibility of knowledge content.** The knowledge carried by an instruction event can be any piece of any of the types of knowledge that the agent uses, provided the knowledge is applicable in some way within the ongoing task and discourse context.
3. **Situation flexibility.** An instruction event can apply either to the current task situation, or to some specified hypothetical situation.

Unfortunately, like the mapping problem, the general interaction problem is extremely difficult, because the space of possible instruction events is enormous. This work does not present a complete solution; however, by targeting the dimensions of flexible knowledge content and situation flexibility, a broader range of interaction capabilities than previous systems is achieved. The following sections discuss each dimension of the interaction problem in more detail.

#### Flexible initiation

In human tutorial dialogues, initiation of instruction is mixed between student and teacher. One study indicates that teacher initiation is more prevalent early in instruction; student initiation increases as the student learns more, and then drops off again, as the student masters the task [Emihovich and Miller, 1988]. Teacher initiation may dominate early on because the student has so little knowledge that the teacher assumes instruction will be constantly needed.

Supporting teacher-initiated instruction events is difficult because the instruction event interrupts other ongoing processing that the agent may be engaged in. Upon interrupting the agent, an instruction event may alter the agent's knowledge in a way that could change or invalidate the reasoning the agent was previously engaged in. Because of these difficulties, instructable systems, learning apprentice systems, and interactive knowledge acquisition tools to date have not supported fully instructor-initiated instruction events. Likewise, in this work, instructor-initiated instruction events are not supported.

Agent-initiated instruction can be directed in (at least) two possible ways: verification driven or impasse driven. Some learning apprentice systems, such as LEAP [Mitchell *et al.*, 1990] and DISCIPLE [Kodratoff and Tecuci, 1987b] ask the instructor to verify or alter each reasoning step. The advantage of this approach is that each step is examined by the instructor; the disadvantage, of course, is that each step *must* be examined.

An alternative approach is to drive instruction requests by impasses in the agent's task performance [Golding *et al.*, 1987; Laird *et al.*, 1990]. This is the approach followed here. An impasse in task performance indicates that the agent's knowledge is lacking and it needs instruction. The advantage of this approach is that as the agent learns, it becomes more autonomous; its need for instruction decreases over time. The disadvantage is that not all lacks of knowledge can be recognized by reaching impasses; for example, an impasse will not occur when a task solution is correct but inefficient. The agent described here also asks for verification when it attempts to induce new knowledge (not verification of performance, but of learning).

### Flexibility of knowledge content

A complete tutorable agent must handle instruction events involving any knowledge that is applicable in some way within the ongoing task and discourse context. Similar to the notion of discourse coherence [Mann and Thompson, 1988], a complete tutorable agent needs to support any instruction event with *knowledge* coherence; that is, any instruction event delivering knowledge that makes sense in current context. This requirement is difficult to meet in general, because of the wide range of knowledge that may be considered relevant to any particular situation (e.g., obscure analogies). Accepting any utterance the instructor might make at any point requires a powerful ability to relate each utterance to the ongoing discourse and task situation. No instructable systems have met this requirement fully.

However, Instructo-Soar meets the flexible knowledge content requirement for one particular class of instructions: action specifications (i.e., imperative commands). This type of instruction is especially prevalent in tutorial instruction for procedures. Supporting flexible knowledge content for this class of instruction means that the instructor is free to give any command at any point during task performance, without concern for whether the agent knows how to perform the command yet or not.

For any given 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 (extra unknown steps are required); or (3) the commanded action is unknown. Thus, supporting flexible knowledge content for commands means that while leading the agent through a task, the instructor may skip steps (possibility 2) or command unknown subtasks (possibility 3) at any point. If the agent knows how to achieve any skipped steps and the commanded action, it will do so. If not, it will reach an impasse and ask for more instruction.

The theory of tutorial instruction described here assumes that when an imperative command is given in response to a task impasse, the command applies to the most recent unachieved goal, as opposed to some higher level goal or an unrelated event. This constraint, called the *one-goal-at-a-time* constraint, is assumed as a *felicity condition* of tutorial instruction; that is, a communication convention that is naturally followed [VanLehn, 1987]. The interaction pattern that results, in which procedures are commanded and then taught as needed, has been observed in human tutorial instruction. As Wertsch [1979] observes: "...adults spontaneously follow a communication strategy in which they use directives that children do not understand and then guide the children through the behaviors necessary to carry out these directives."

The example in Table 1.2 illustrates commanding unknown actions and skipping steps. The first command, "Pick up the red block," is a completely unknown procedure to the agent, and further instruction is requested. This further instruction may include other completely unknown tasks (an example of this is given in Appendix A). In this case, while teaching "pick up", the instructor asks the agent to "Move the arm above the red block." This procedure is known, but has the precondition that the arm must be raised. However, the arm is lowered; the instructor has skipped the step of moving the hand up. After being told to "Move up," the agent is able to move

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.

Table 1.2: Instructions for "Pick up."

above the red block. Notice the non-canonical sequence of instructions that results: "Move down" immediately follows "Move up" in the instruction sequence, but not in execution.

Supporting flexible interaction for commands gives the instructor a great amount of flexibility in teaching a set of tasks. Appendix B presents a mathematical analysis of the number of possible sequences of instructions that can be used to teach a given procedure. The number grows exponentially with the number of actions that must be executed to perform the procedure. For a procedure with six actions, there are over 100 possible instruction sequences; for seven, the number grows to over 400.

The implemented agent handles other types of instructions besides action commands at appropriate points in the ongoing dialogue between the agent and the instructor, but not in a completely flexible way, as described in Chapters 3 and 4.

### Situation flexibility

A complete tutorable agent must handle instructions that apply to whatever situation the instructor pleases, be it the current task situation, or some hypothetical situation that the instructor specifies. Instructors make frequent use of both of these options. Analysis of a protocol of a student being taught to use a flight simulator revealed that 119 out of 508 instructions (23%) involved hypothetical situations, with the remainder applying to the current situation at the time they were given.

Instructions that apply to the current situation are called *implicitly situated* [Huffman and Laird, 1992]; the situation they are intended to apply to is implicit rather than explicitly stated. Since the instruction itself says nothing about the situation it should be applied in, the current situation (the task being performed and the current state) is implied.

As an example, consider the following exchange from a protocol of an instructor teaching a simulated robot:

> Pick up the block.
> *I don't know how.*
>> To start, you need to get the robot arm over the block.
>> Now, put your hand on top of the block and grab it.
>> Then, lift again.
>> That's the end of 'pick up the block.'

Each of these instructions is implicitly situated. For instance, "Then, lift again" applies to the higher level task "Pick up the block," in the state where the arm is holding the block and the block is on the table.

In contrast, instructions that specify elements of the situation they are meant to apply to are *explicitly situated* [Huffman and Laird, 1992]. The language of the instruction explicitly indicates a hypothetical situation. The agent is not meant to carry out the instruction immediately (as an implicitly situated instruction), but rather when a situation arises that is like the one specified by the instruction.

A simple example is:

> *Never* touch a hot stove.

Here, the language indicates that the instruction applies whenever there is a hot stove present, no matter what other tasks are being performed. Other explicitly situated instructions include conditionals and instructions with purpose clauses [DiEugenio, 1992], such as:

> *For normal tape*, employ the SF/NORM position.
> *When using chocolate chips*, add them to coconut mixture just before pressing into pie pan.
> *To restart this*, you can hit R or shift-R.

As a number of researchers have pointed out [Ford and Thompson, 1986; Johnson-Laird, 1986; Haiman, 1978], conditional clauses introduce a shared reference between speaker and hearer that forms an explicit background for interpreting or evaluating the consequent.[1] Here, the clauses in italics indicate a hypothetical situation that the command in the remainder of the instruction is meant to apply to. In most cases, the situation is only partially specified, with the remainder drawn from the current situation, as in "When using chocolate chips [and cooking this recipe, and at the current point in the process]..."

Explicitly situated instructions allow the instructor to indicate contingencies while teaching a particular case of a procedure, or to describe courses of action for situations that are rare or dangerous, and thus would be difficult to teach using implicitly situated instructions. Explicitly situated instructions are also useful for "going back" to teach about a situation that has already passed by. This can be important in dynamic domains where the situation is continually changing.

In general, a hypothetical situation may be created and referred to across *multiple* utterances. A complete tutorable agent must be able to handle this possibility. An example taken from the flight simulator protocol is shown in Table 1.3. This entire sequence of instructions applies to a series of related hypothetical situations, in which the plane has taken off and the flaps must be lowered. None of the instructions in the sequence is actually carried out in the real world when it is given.

The theory presented here covers both implicitly and explicitly situated instructions, as described in Chapter 2. However, the implemented agent's situation flexibility is limited to single instructions; hypothetical situations that exist through multiple utterances are not handled. Instructo-Soar's performance on implicitly situated instructions is described in Chapters 3, and on explicitly situated instructions in Chapter 4.

Table 1.4 summarizes the requirements of the interaction problem.

### 1.2.3 The transfer problem

The transfer problem is the problem of learning generally applicable knowledge from instructions. This general knowledge will transfer to appropriate situations that arise in the future. Solving this problem involves more than simply memorizing the instructions; rather, the proper conditions of applicability of each instruction must be determined.

For example, consider the following exchange between an instructor and an agent:

> Block open our office door.
> *How do I do that?*
> Pick up a red block.
> *OK.*
> Now, drop it here, next to the door.

[1] Some types of conditionals do not follow this pattern [Akatsuka, 1986], but they are not relevant to tutorial instruction.

| | Speaker | Instruction |
|---|---|---|
| 19 | Instructor | You have some flaps, which are control surfaces on your wings |
| 20 | | that help you take off and land. |
| 21 | | And these are initially set up. |
| 22 | | You'll have to get rid of them once you've taken off, so... |
| 23 | Student | So you have them up to take off? |
| 24 | Instructor | They are up initially to take off |
| 25 | | so you want to...you want to decrease them all the way to zero degrees... |
| 26 | Student | Once you get off the ground. |
| 27 | Instructor | Once you get off the ground, yeah. |
| 28 | | Each plane has a particular range that their flaps can go, |
| 29 | | and so you want to decrease them all the way, |
| 30 | | so you hit shift-F. |
| 31 | | Each capital F decreases flaps by 10 degrees |
| 32 | | so you'll just keep hitting that... |
| 33 | Student | So you've got to hit it a few times. |
| 34 | Instructor | Yeah...and on the fighters it's like 30 degrees, |
| 35 | | so you have to hit it three times to get rid of it. |

Table 1.3: An example of a hypothetical situation existing through multiple instructions.

| | Requirement | Met in this work? |
|---|---|---|
| 1. | Flexible initiation of instruction | no (only impasse-driven agent-initiated) |
| 2. | Flexibility of instructional knowledge content | partial (full flexibility for action commands) |
| 3. | Situation flexibility | partial (both implicitly and explicitly situated instructions, but no multiple utterance hypotheticals) |

Table 1.4: The requirements of the interaction problem.

What are the proper applicability conditions for the instruction "Pick up a red block"? Simple memorization would yield poor (low transfer) learning; something like *whenever blocking open an office door, pick up a red block.* However, the fact that the block is red, and even that it is a block, are irrelevant in this case. Rather, the fact that the block weighs (say) over five pounds, giving it enough friction with the floor to hold open the door, is crucial. The proper learning is something like:

**If** trying to block open a door, and
there is an object **obj** that is pick-uppable, and
**obj** weighs more than 5 pounds
**then** the best thing to do next is pick up **obj**.

Solving the transfer problem involves both generalization and specialization. Here, the original instruction is both generalized (`color red` and `isa block` drop out) and specialized (`weight > 5` is added).

The transfer problem places a number of demands on a complete tutorable agent:

1. **General learning from specific cases.** The agent is instructed to perform a task in a particular situation, but is expected to learn generally to perform the task in sufficiently similar situations.

2. **Fast learning.** An instructable agent is expected to learn new procedures quickly. Each task should only have to be taught once.

3. **Maximal use of prior knowledge.** An agent must apply its prior knowledge in learning from instruction. This is a maxim for machine learning systems in general (if you have knowledge, use it), and is particularly relevant for learning from instruction because learning is expected to happen quickly.

4. **Incremental learning.** The agent must be able to continually increase in knowledge through instruction over the course of its lifetime. New knowledge must be smoothly integrated into the existing knowledge base as a part of the agent's ongoing performance.

5. **Learning of all types of knowledge used in task performance.** An agent uses multiple types of knowledge in performing a task; control knowledge, knowledge about objects and their properties, knowledge of the effects of actions, etc. The agent may be missing pieces of knowledge of any of these types. Thus, a complete tutorable agent should be able to learn any of these types of knowledge from instruction. The theory of learning from tutorial instruction presented in Chapter 2 makes this a testable criterion, by laying out the types of knowledge in an agent based on a particular computational model of intelligent agents.

6. **Dealing with incorrect knowledge.** The agent's knowledge is clearly incomplete; otherwise, it would not need instruction. However, its knowledge may also be incorrect. Incorrect knowledge may be a part of the agent's original knowledge, or may be learned from incorrect instruction. A complete tutorable agent must be able to perform and learn effectively in the face of incorrect knowledge.

7. **Learning from instruction coexisting with learning from other sources.** In addition to instruction, a complete agent should be able to learn from other sources of knowledge that are available [Huffman *et al.*, 1993b], such as external observation, analogy to other domains, etc. These other types of learning may interact with instructional learning. Instructions may suggest their use (e.g., "Try this out and notice what happens").

The theory of learning from tutorial instruction presented here allows the requirements (1) through (5) of this list to be met. The research has focused on adding to incomplete knowledge through instruction, and does not explicitly deal with the issue of incorrect knowledge (requirement 6). The instructional learning method that has been developed should be readily combinable with

| | Requirement | Met in this work? |
|---|---|---|
| 1. | General learning from specific cases | yes (via *situated explanation*) |
| 2. | Fast learning | yes (new procedures only taught once) |
| 3. | Maximal use of prior knowledge | yes |
| 4. | Incremental learning | yes |
| 5. | Learning all types of knowledge used in task performance | yes (all knowledge types of the *problem space computational model* [Newell *et al.*, 1990]) |
| 6. | Dealing with incorrect knowledge | no (only extending incomplete knowledge) |
| 7. | Coexistence with learning from other sources | no (not demonstrated) |

Table 1.5: The requirements of the transfer problem.

learning from other sources (requirement 7), but as the focus here is on instruction, this has not been demonstrated.

The requirements of the transfer problem are summarized in Table 1.5. The most distinguishing feature of Instructo-Soar is its ability to learn each of the types of knowledge it uses during task performance from instruction (requirement 5). Thus, it can be termed *complete* with respect to instructability of its knowledge types.

## 1.3 Related work

### 1.3.1 Artificial Intelligence

A number of systems have been produced that behave in response to natural language instructions, but do not do extensive learning. One of the earliest and most famous is SHRDLU [Winograd, 1972]. SHRDLU plans and acts in a blocks-world domain in response to natural language commands. The focus is primarily on planning and on the mapping problem; SHRDLU transforms natural language input into code fragments that are run to produce the instructed action. The system does a small amount of rote learning, such as learning new goal specifications by directly transforming sentences into state descriptions, but does not learn how to perform procedures.

More recent research has investigated the mapping problem for more powerful agents. Chapman's [1990] SONJA plays a video game and can take instructions during the course of the action; the system uses the pragmatics of the ongoing task to understand incompletely specified instructions. The AminNL project [DiEugenio and White, 1992; DiEugenio and Webber, 1992; Kalita, 1991; Kalita and Badler, 1990] focuses on the mapping problem for an animated agent. Neither of these systems do any learning.

One of the most impressive recent agents that behaves in response to natural language is Vere and Bickmore's [1990] Homer. Homer lives in a simulated sea world, can do complex temporal planning, and can understand a range of instructions, focusing on instructions concerning actions to be taken in a timely manner. Homer does some simple learning; for instance, it can learn from assertions ("You own the log"), and from conditional instructions indicating actions to be taken later ("If you see a bird, photograph it"). It also has a detailed episodic memory, allowing the instructor to refer to past events and instructions, and to ask for explanations of actions. However, Homer does not learn procedures or other kinds of domain knowledge from sequences of instruction, or make use of prior knowledge to produce general learning.

Learning from tutorial instruction is related to work on learning from external guidance and advice taking [McCarthy, 1968; Rychener, 1983; Carbonell *et al.*, 1983]. Some early systems learned declarative, ontological knowledge from instructions [Lindsay, 1963; Haas and Hendrix, 1983]. UNDERSTAND [Simon, 1977; Simon and Hayes, 1976] was among the first to do procedural learning from instruction. It took a natural language description of a task that was isomorphic to the Tower of Hanoi problem, and built GPS-style operators and difference tables to solve the task.

Mostow's FOO [Mostow, 1983a; Mostow, 1981; Hayes-Roth *et al.*, 1981] and BAR [Mostow, 1983b] were early attempts to build a system that could learn from advice. FOO and BAR take very general advice, such as "Avoid taking points" for the card game Hearts (the actual input is not in natural language). The advice is unsituated and non-interactive; it is not given during performance of any specific task. It is operationalized by applying a sequence of hand-selected transformations. Because the advice is unsituated and can potentially apply to many different situations, it is difficult to learn from. "Avoid taking points" requires a large number of transformations, producing multiple operationalized search heuristics for playing Hearts. Such advice would be of little use to a human learner, because the unsituated operationalization problem is so difficult. Thus, the focus of the work in this thesis is situated instruction.

Recently, some other researchers have begun to focus specifically on the issue of learning from situated natural language instruction. Martin and Firby [1991] discuss an approach to interpreting and learning from elliptical instructions ("Use the shovel") by matching the instruction to expectations generated from a task execution failure. Alterman and his colleagues [Alterman *et al.*, 1991; Alterman and Carpenter, 1991] have produced FLOBN, a system that learns to operate a device by using instruction and analogy to a previously known device. FLOBN makes use of written "recipe-like" instruction that specifies a sequence of steps to achieve a goal, rather than the type of interactive instruction that this work is concerned with. It learns by relating instructed steps for a procedure to previously known procedures with similar goals.

Most other work on learning from instruction-like input has been under the rubric of learning apprentice systems (LAS's). Learning apprentice systems have been defined as "interactive, knowledge-based consultants that directly assimilate new knowledge by observing and analyzing the problem-solving steps contributed by their users through their normal use of the system" [Mitchell *et al.*, 1990]. Tutorial instruction extends past work in LAS's in two directions: interaction flexibility and types of knowledge learned. Past LAS's have interacted by either observing an expert's solution to specific problems [Redmond, 1992; Mitchell *et al.*, 1990; Wilkins, 1990; Segre, 1987; VanLehn, 1987] or by attempting to solve problems and allowing the expert to verify and critique decisions as they are made by the performance system [Porter *et al.*, 1990; Laird *et al.*, 1990; Gruber, 1989; Kodratoff and Tecuci, 1987b; Golding *et al.*, 1987; Porter and Kibler, 1986]. LAS's that learn by observing expert solutions do not provide flexible interaction with the instructor; the "instructions" (the observed actions) are each primitive actions that occur in their canonical execution order to achieve the desired goals. LAS's that allow the expert to critique decisions typically suggest steps to the instructor to be verified or rejected for other actions. Tutorial instruction increases this interaction ability in a number of ways. First, the instructor may specify unknown subtasks, or subtasks with unachieved preconditions, to the agent at any point, even during teaching of larger task. Past LAS's typically do not allow specification of completely unknown subtasks. Second, tutorial instruction allows the use of explicitly situated instructions to teach about contingencies that may not be present in the current situation.

LAS's have used both analytic and inductive learning methods to learn in a variety of domains. However, specific LAS's have focussed on learning particular types of knowledge: e.g., operator implementations (LEAP [Mitchell *et al.*, 1990]), goal decomposition rules (DISCIPLE [Kodratoff and Tecuci, 1987b]), operational versions of abstract (functional) goals (ARMS [Segre, 1987]), control knowledge and control features (ASK [Gruber, 1989]), procedure schemas (a combination of goal decomposition and control knowledge; learned by SIERRA [VanLehn, 1987]), and heuristic classification knowledge (PROTOS [Porter *et al.*, 1990], ODYSSEUS [Wilkins, 1990]). A complete tutorable agent should be able to learn any of the types of knowledge it uses for task performance via instruction. One contribution of this work is making this requirement explicit, and showing how it can be evaluated by delineating the computational model underlying an instructable system's performance. Instructo-Soar meets the requirement, demonstrating a wider breadth of learning than previous learning apprentice systems.

Other interactive knowledge acquisition tools besides learning apprentices are more distantly related to tutorial instruction. Early interactive tools, such as TIERESIAS [Davis, 1979], required direct manipulation of symbol level structures used by the knowledge-based system, instead of allowing interaction at the knowledge level. Thus, the user had to understand the internal workings of the

system. More recent work has focused on method-specific elicitation techniques, based on McDermott's idea of knowledge roles [McDermott, 1988] and Chandrasekaran's generic tasks framework [Chandrasekaran, 1986]. These knowledge-level techniques gain their power by limiting application to a specific problem solving method, with a pre-defined set of knowledge roles [Birmingham and Klinker, 1993]. Method-based knowledge acquisition tools differ from tutorial instruction in that elicitation is unsituated and focused on a particular class of tasks. The tools are applicable to tasks requiring global control strategies based on their built in problem solving method; thus, they need not acquire abstract procedural ("process") knowledge. They acquire knowledge interactively, but the interaction concerns filling of knowledge roles ("data"), rather than situated instruction for specific tasks ("data" and "process" for specific tasks). Tutorial instruction is not limited to a particular problem solving method, but is not easily applied to tasks requiring non-local control.

### 1.3.2 Cognitive Science

Few cognitive studies have examined the course of learning for individual subjects as they are given situated, interactive procedural instruction. Most instructional studies have either examined the effects of various linguistic forms of individual instructions (e.g., [Jones, 1966; File and Jew, 1973; Just and Carpenter, 1976; Wright and Wilcox, 1979; Dixon, 1982]), or the learning effectiveness, measured in terms of transfer, of different global instructional and training strategies (see [Singley and Anderson, 1989] for a review). These studies have shown that one-to-one tutoring is a highly effective teaching technique [Bloom, 1984; Palinscar, 1986]. It robustly produces stronger learning than classroom instruction by two standard deviations [Bloom, 1984].

A number of studies have shown that students' amount of prior knowledge has a large impact on learning from instruction. These results are relevant to the model of learning from tutorial instruction presented here because the model makes heavy use of prior knowledge to leverage learning. For instance, in Kieras and Bovair's [1984] study, subjects were given instructions to operate a device. Subjects given prior knowledge of the device performed much better than those without such knowledge, in terms of speed of performance, accuracy, and retention.

A related result involves the self-explanation effect [Chi and VanLehn, 1991; Chi *et al.*, 1989]. Subjects who attempt to self-explain examples of problem solutions while studying them achieve higher performance when solving problems later than subjects who do not. The theory of learning from tutorial instruction presented here is based on a self-explanation process, as described in the next chapter. Although receiving tutorial instruction is somewhat different from studying pre-solved examples, the prediction is the same; namely, that self-explanation allows greater learning and transfer.

Finally, this work can be compared with other cognitive models of procedural learning. Most recent models are based on Anderson's model of procedural learning in ACT* [Anderson, 1983; Anderson, 1987; Singley and Anderson, 1989], which in turn is based on Thorndike's [1903] theory of identical elements. Thorndike proposed that all transfer occurred because of the use of identical elements – specific stimulus-response rules – across multiple tasks. In Anderson's and other related theories (including the one presented here), the identical elements are production rules.

Chapter 5 gives further discussion of relevant human behavioral data and related cognitive models, after describing the behavioral properties and predictions of the theory presented here.

## 1.4 Outline of the rest of the thesis

This chapter has presented a definition and task analysis of the problem of learning from tutorial instruction. It has described the three problems facing an instructable agent: the mapping problem, the interaction problem, and the transfer problem. Any instructable agent must solve each of these problems to some degree. As a metric for performance and comparison of instructable agents, the requirements of each problem for a *complete tutorable agent* were developed. The focus of this work is on the interaction and transfer problems.

The rest of the thesis is organized as follows. Chapter 2 presents the theory of learning from tutorial instruction in abstract terms. The theory includes a computational model of an agent, the

*problem space computational model*, and a learning technique, *situated explanation*. The computational model defines the learning task by identifying the types of knowledge of an agent; each of these types of knowledge should be learnable through instruction. The learning technique specifies how an instructable agent can learn this knowledge from interactive instructions.

Chapters 3 and 4 describe an implemented instructable agent, Instructo-Soar, that embodies the theory of Chapter 2. Instructo-Soar is able to learn each of the types of knowledge it uses in task performance from instruction, and can handle both implicitly and explicitly situated instructions, and the range of interaction flexibility described earlier. Chapter 3 describes Instructo-Soar's learning of completely new procedures, and extension of known procedures to new situations, through implicitly situated instruction. Describing these capabilities will make the agent's basic learning and task performance clear. Chapter 4 describes Instructo-Soar's learning of each of the remaining types of knowledge, and its handling of explicitly situated instructions.

Instructo-Soar's possible application as a cognitive model is explored in Chapter 5. The theory has been strongly influenced by its development within the Soar architecture and theory of cognition. The chapter analyzes these influences, with an eye towards teasing apart the influence of various aspects of Soar on the different behavioral properties of the instructional learning model. Based on these behavioral properties, a set of behavioral predictions is generated, and the model and its predictions and compared with known human behavioral data and other cognitive models of procedural learning.

In Chapter 6, the results of the theory and implementation are analyzed. The current limitations of the work and opportunities for future research are discussed in detail. Chapter 7 concludes with a review of the major contributions of the thesis.

Appendix A describes Instructo-Soar's performance on a large instructional example that highlights each of the system's major capabilities. Appendix B gives a mathematical analysis of the number of instruction sequences that can be used to teach a given procedure. This analysis indicates the high degree of flexibility that an instructable agent like Instructo-Soar provides to its instructor. Appendix C describes the details of how Soar's chunking mechanism produces general rules from Instructo-Soar's situated explanation process.

# Chapter 2

# A Theory of Learning from Tutorial Instruction

This chapter presents a general theory of learning from tutorial instruction. The theory has two parts. First, a computational model of an intelligent agent, known as the *problem space computational model (PSCM)* [Newell *et al.*, 1990; Yost, 1993] specifies the general form of the agent's processing, indicating each of the types of knowledge that the agent has and how they affect its performance. Specifying these types of knowledge defines the tutorial learning task and provides an evaluation metric for an instructable agent: it should be able to learn each of its knowledge types from instruction. Second, a learning technique, called *situated explanation* [Huffman and Laird, 1994], specifies how an instructable agent can derive general knowledge from the instructions it is given. The notion of explanation that is employed allows for both deductive and inductive steps, and supports interactive instruction by allowing multiple, embedded levels of explanation and instruction taking. The implemented instructable agent that embodies the theory is described in Chapters 3 and 4.

## 2.1 The problem space computational model (PSCM)

A computational model is a "description of a class of systems in terms of a set of operations on entities that can be interpreted in computational terms" [Newell *et al.*, 1990, p. 6]. Specifying the computational model of an agent defines the complete set of "operations on entities" that the agent carries out during its performance. Each operation must be performed with knowledge; thus, defining the basic operations of an agent's computational model defines the complete set of types of knowledge that the agent uses in task performance.

A computational model for a general instructable agent must meet two requirements:

1. **Support of general computation.** For a general agent, any type of computation might be required. The computational model must allow this flexibility.

2. **Close correspondence to knowledge level components.** The components of the computational model – the "operations" and "entities" – must correspond closely to the knowledge level [Newell, 1981]. This is required because the goal in identifying the computational model is to elucidate classes of knowledge that instructions map to and that are learned. Since tutorial instructions provide knowledge at the knowledge level (see Section 1.1), the further away the components of the computational model are from the knowledge level, the more difficult mapping and learning from instructions will be.

One class of computational models is standard programming languages (e.g., Lisp). These languages clearly support general computation. However, their operations and constructs are at the symbol level, without direct correspondence to the knowledge level. For example, dereferencing a pointer does not directly correspond to any particular knowledge level action. Thus, using a symbol level computational model would not provide much leverage in elucidating agent knowledge types. Rather, an additional theory of mapping from instructions to the symbol level components of these

computational models would be required. The same problem exists for theoretical computational models, such as Turing machines and push-down automata.

Similarly, connectionist and neural network models of computation (e.g., [Rumelhart and McClelland, 1986]) employ (by design) computational "operations and entities" that are at a level far below the knowledge level. Thus, these models are not appropriate as the top-level computational model for an instructable agent. However, because higher levels of description of a computational system are implemented by lower levels [Newell, 1990], these models of computation might be used as the implementation substrate for the higher level computational model of an instructable agent.

Logic is another possible alternative. The entities within logics (e.g., propositions, well-formed formulas) are well matched to the knowledge level. Logics also specify the set of legal operations on entities (e.g., modus ponens). The combination of logical formulas and logical operations define the space of what can *possibly* be computed. However, logic provides no notion of what *should* be computed; that is, although logics provide computational operations, they do not specify the *control* of of those operations' application. Implemented logic systems (e.g., PROLOG) must add extra control mechanisms that are outside of the logical formalism (e.g., the PROLOG "cut"). Control knowledge is a crucial type of knowledge in actual agents, that can be communicated by instructions. Thus, it is desirable for the computational model of an instructable agent to include control knowledge.

Another class of computational models with knowledge level components are the problem solving methods that have been defined in the knowledge acquisition community; for example, heuristic classification [Clancy, 1985], propose-and-refine constraint satisfaction [Marcus and McDermott, 1989], and the various generic tasks methods [Chandrasekaran, 1986]. These methods specify the knowledge they require in terms of knowledge roles [McDermott, 1988] within a pre-defined control structure. Because they are designed for particular families of tasks, these methods do not support general computation in a flexible way, and thus are not an appropriate choice for the computational model of a general instructable agent. However, these methods are appropriate as computational models for instructable systems targeted at a particular type of problem; PROTOS [Porter *et al.*, 1990], a learning apprentice for heuristic classification tasks, is an excellent example.

Since the goal of specifying an agent's computational model is to specify the agent's knowledge types, it might appear that selecting a theory of knowledge representation, instead of a computational model, would be a viable alternative. Such theories define the functions and structures that are to be used in representing knowledge of various kinds (e.g., KL-ONE [Brachman, 1980]); in addition, some define the possible content of those structures (e.g., conceptual dependency theory [Schank, 1975], CYC [Guha and Lenat, 1990]). However, since these theories do not specify computational operations, they do not indicate how procedural knowledge is used within an agent. Computational structure must be added to these theories to produce working agents.

Rather than an *alternative* to specifying a computational model, a theory of knowledge representation is an *addition*, specifying how the knowledge contained in the computational entities and operations of the computational model is represented. A complete content theory of knowledge representation would provide a more fine-grained breakdown of the knowledge to be learned within each category specified by the computational model. Defining the range of knowledge that can be represented is especially important with respect to the mapping problem, since these possible representations form the range of the mapping (see Section 1.2.1). As the mapping problem is not the focus of this work, a content theory of knowledge representation is not included as a part of the theory developed here.

The computational model adopted here is called the problem space computational model (PSCM) [Yost, 1993; Newell *et al.*, 1990; Yost and Newell, 1989]. The PSCM allows us to describe an agent's structure in a particular computational framework – computation within problem spaces – without reference to the symbol level structures the agent uses to implement the computation. The "operations and entities" within the PSCM are at a level above the symbol level, and approximating the knowledge level [Newell *et al.*, 1990], thus meeting the requirement of close correspondence to knowledge level components. Soar [Laird *et al.*, 1987] is one symbol level implementation of the PSCM, and has been selected as the platform in which to implement the instructable agent described here.

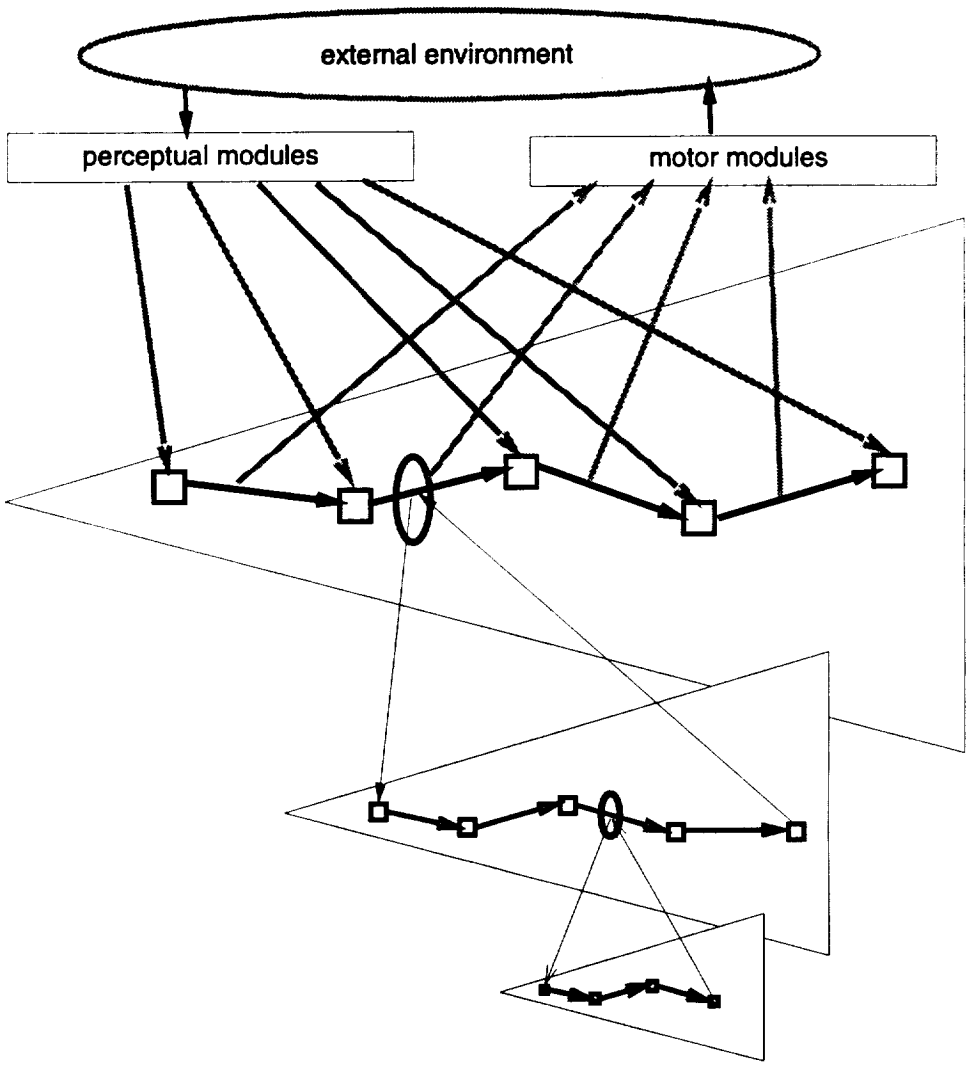



Figure 2.1: The processing of an agent within the problem space computational model. Triangles represent problem spaces; squares, states; arrows, operators; and ovals, impasses.

A schematic of a PSCM agent is shown in Figure 2.1. Perception and motor modules connect the agent to the external environment. A PSCM agent reaches a goal by moving through a sequence of *states* in a *problem space*. A state consists of objects and attributes that reflect the current point in processing. A PSCM agent makes progress towards its goals by sequentially applying *operators* to the current state. Operators are essentially functions that produce a new state from the current state, either by modifying the internal state representation or causing an action in the external world. The PSCM 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 (termed a *subgoal*) is created, with the goal of resolving the impasse. Operators are selected and applied to states within this second context; another impasse may be reached, causing a third context to arise, and so on.

In the PSCM, the only computational entities mediated by the agent's knowledge are states and operators.[1] Thus, there are a small set of basic PSCM-level functions the agent must perform using its knowledge:

1. **State inference.** The agent must construct a representation of the current state it is in. This involves comprehending information delivered by perceptual modules, and making inferences about state properties based on other state properties. For instance, an agent might have the knowledge that a block is held if the gripper is directly above it and closed.

2. **Operator selection.** The agent must select an operator to apply to the current state. This involves two types of knowledge:

   2.1. *Proposal knowledge.* Proposals indicate operators that are deemed appropriate for the current situation; e.g., that `close-hand` is appropriate when the hand is above a block the agent is trying to grasp.

   2.2. *Preferential knowledge.* Preferential knowledge compares and orders proposed operators; e.g., "A is better than B"; "A and B are equally good"; "C is best"; "D is rejected."

3. **Operator application.** Once selected, an operator is applied to make changes to the current state, and perhaps produce motor commands. An operator may be applied directly, or indirectly via a subgoal. Thus, operator application knowledge can take two different forms:

   3.1. *Operator effects.* The operator is applied directly in the current problem space. The agent has knowledge of the effects of the operator on the state and motor commands produced (if any).

   3.2. *Sub-operator selection.* The operator is applied by reaching an impasse and selecting a sequence of operators in a subgoal. Knowledge of the operator's application in this case equals knowledge of the selection of the sub-operators. Thus, this is equivalent to operator selection knowledge (2, above).

4. **Operator termination.** An operator must be terminated when its actions have been completed. The termination conditions, or *goal concept* [Mitchell *et al.*, 1986], of an operator indicate the state conditions that the operator is meant to achieve. For example, the termination conditions of `pick-up(blk)` might be that `blk` is held and the arm is raised.

Each of these functions is performed with knowledge. Thus, they define the set of knowledge types present within a PSCM agent. The knowledge types are summarized in Table 2.1.[2]

Because of the recursive nature of the PSCM, what can be thought of as the "goals" to be achieved in a particular subgoal are typically PSCM *operators* in higher subgoals that must be

[1] This version of the PSCM is slightly different than that presented in [Yost and Newell, 1989] and [Newell *et al.*, 1990]. The main difference is that those PSCM descriptions included "task formulation" (selecting and specifying a task to perform) as a basic operation, whereas here it is assumed that task formulation can be done by operators.

[2] These knowledge types are more general than the typical knowledge roles for particular problem solving methods [Balkany *et al.*, 1991], such as heuristic classification or propose-and-revise. They might be thought of as the knowledge roles for the PSCM, but the PSCM is more general than such problem solving methods.

| Entity | Knowledge type | Example |
|---|---|---|
| state | inference | If the gripper is closed and directly above a block, then the gripper is holding the block. |
| operator | proposal | If the goal is to pick up block1, and block1 is on table1, and the robot is not docked at table1, then propose moving to table1. |
| operator | preferential | (1) If picking up an object is proposed, and the object is explosive, then reject picking it up.<br>(2) if the goal is to pick up a small metal object on table1, prefer moving to table1 over picking up the magnet. |
| operator | effects | An effect of the operator move to table1 is that the robot becomes docked at table1. |
| operator | termination | The termination conditions of the operator "pick up block1" are: the gripper is holding block1 and the gripper is raised. |

Table 2.1: The five types of knowledge of a PSCM agent.

completed. For example, if an agent selects an operator to pick up a block, and it cannot be directly applied, a subgoal will be created; the "goal" of this subgoal is to complete the pick up operator.

The PSCM is a broad formulation that supports general computation. Newell [1980, 1990] has gone so far as to propose problem spaces as the basis for general intelligent behavior (the *problem space hypothesis*). The PSCM supports a full range of behavior from search based knowledge lean behavior (any weak method can be built in PSCM [Laird, 1983]) to knowledge rich behavior that does not require search. A least commitment, local type of control is supported, because operators are selected sequentially based on local information. This type of control is well matched to the kinds of tasks most readily taught by tutorial instruction, as discussed in Chapter 1. However, local control decisions in the PSCM can be combined to exhibit global control behavior, as demonstrated by Yost's work on TAQL [Yost, 1993; Yost and Newell, 1989], a PSCM-based knowledge acquisition language that acquired tasks requiring global control.

Finally, the PSCM supports incremental, monotonic extensions of an agent's knowledge. Because the PSCM applies each piece of knowledge locally and independently, new pieces of knowledge of any type can be added without requiring reformulation of prior knowledge. This is an important property for an instructable agent, which must incrementally learn new knowledge from instruction over the course of its lifetime. If new knowledge conflicts with existing knowledge, the conflict will not be fatal but will result in an impasse during later performance, allowing the agent to do further reasoning (and possibly receive further instruction) to resolve the conflict.

Specifying the computational model has clarified the task faced by an instructable agent. The agent must be able to learn each of the five types of knowledge specified by the PSCM from instruction. The next section discusses the learning process itself.

## 2.2 Learning by situated explanation

Learning from tutorial instruction requires both analytic learning (based on prior knowledge), and inductive learning (when prior knowledge is insufficient). The goal of this work is not to produce a more powerful analytic technique or a better inductive method. Rather, the goal of the theory is to specify how these techniques come together to apply to the comprehensive learning task of tutorial instruction.

### 2.2.1 Overview

As described in Chapter 1, a theory of learning from tutorial instruction must meet a number of requirements. The goal of this theory is learning that meets the requirements of the transfer problem (Section 1.2.3) associated with extending incomplete knowledge through instruction, while

| | Requirement | From |
|---|---|---|
| 1. | General learning from specific cases | Transfer |
| 2. | Fast learning (each task instructed only once) | Transfer |
| 3. | Maximal use of prior knowledge | Transfer |
| 4. | Incremental learning | Transfer |
| 5. | Ability to learn all types of knowledge used in task performance | Transfer |
| 6. | Agent-initiated instruction | Interaction |
| 7. | Flexibility of knowledge content: full flexibility for action commands; others as appropriate | Interaction |
| 8. | Situation flexibility: both implicitly and explicitly situated instructions | Interaction |

Table 2.2: The transfer and interaction requirements of tutorial instruction to be met by the learning theory.

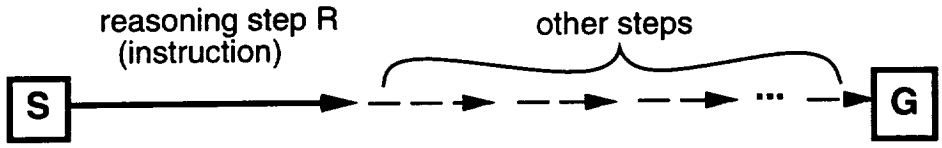



Figure 2.2: Caricature of an explanation of how reasoning step $R$ applies to the situation with the initial state $S$ and the goal $G$ to be achieved.

supporting flexible interaction that meets the targeted requirements of the interaction problem (Section 1.2.2). Providing flexible interaction affects the learning theory because learning must occur in the context of the ongoing interaction between the agent and the instructor. Table 2.2 summarizes the requirements to be met by the theory.

While some of these requirements are useful primarily as evaluation criteria, others place a clear constraint on the type of learning approach that might be used. In particular, requirements one through three specify that general learning must occur from single, specific examples, by making maximal use of prior knowledge. These requirements bode strongly for a learning approach based on *explanation*. The use of explanation to produce general learning has been a common theme in machine learning (e.g., [Fikes *et al.*, 1972; DeJong and Mooney, 1986; Rosenbloom *et al.*, 1988; Schank and Leake, 1989; Minton *et al.*, 1989]; many others) and cognitive science (e.g., [Anderson, 1983; Rosenbloom and Newell, 1986; Lewis, 1988; Chi *et al.*, 1989]). Forming explanations enables general learning from specific cases (requirement 1) because the explanation indicates which features of the case are important; features that do not appear in the explanation may be generalized over. Learning by explaining is fast, typically requiring only a single example (requirement 2), in contrast to correlational induction techniques, which require many examples. Prior knowledge is employed to construct the explanation (requirement 3); the prior knowledge provides the strong bias that allows learning from one example.

To form an explanation of some reasoning step, the way in which the step leads from the current state of reasoning to achievement of a current goal must be determined. An explanation is a path of reasoning from the current state to the goal, through the reasoning step being explained, as diagrammed in Figure 2.2. The figure indicates the three key elements of an explanation: the reasoning step being explained, the endpoints of the explanation (initial state and goal to be reached), and the other reasoning steps required to complete the explanation.

What form do these elements of an explanation take for tutorial instruction?

- *Reasoning step to be explained.* In tutorial instruction, a reasoning step to be explained is an individual instruction given to the agent.

  Alternatively, an entire instruction episode – e.g., the full sequence of instructions for a new procedure – could be explained at once. Applying explanation to single steps results in knowl-

edge applicable at each step (e.g.,[Laird *et al.*, 1989; Golding *et al.*, 1987]); explaining full sequences of reasoning steps results in learning schemas that encode the whole reasoning episode (e.g., [Mooney, 1990; Schank and Leake, 1989; VanLehn, 1987]). Learning factored pieces of knowledge rather than monolithic schemas or plan structures allows the knowledge to be applied in a more reactive fashion, since knowledge is accessed locally based on the current situation [Laird and Rosenbloom, 1990; Drummond, 1989]. This meshes with the PSCM's local control structure. Explaining individual instructions is also supported by psychological results on the self-explanation effect, which have shown that subjects who self-explain instructional examples do so by re-deriving individual lines of the example. "Students virtually never reflect on the overall solution and try to recognize a plan that spans all the lines" [VanLehn and Jones, 1991, p. 111].

- *Endpoints of explanation.* The endpoints of the explanation – the initial state of reasoning and the goal to be achieved – together comprise the *situation* that the instruction applies to. Requirement 8 from Table 2.2 specifies that this situation may be either the current state of the world and goal being pursued, or some hypothetical situation that is specified explicitly by the language of the instruction.

  Most work in machine learning has involved explanations based on the *current* situation of the performance system, in which the goal and state are known. An exception is learning apprentice systems that learn by observing expert behavior, where the goals that observed actions are meant to achieve may not be stated. Determining goals by observing actions is a process of plan recognition [Redmond, 1992; Kautz and Allen, 1986]. In the type of instruction considered here, however, plan recognition is not necessary, because the goal that an instruction is intended to achieve is always either the current goal or a stated, hypothetical goal.

- *Other required reasoning steps.* To complete an explanation of an instruction, an agent must bring its prior knowledge to bear to complete the path through the instruction to achievement of the goal the instruction applies to.

  Early work in explanation-based learning (EBL) [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] required deductive explanations or "proofs" built from a complete and correct domain theory. Tutorial instruction requires a broader notion of explanation, since the point of receiving instruction at all is that the agent's knowledge is incomplete in various ways. An explanation may include "deductive" steps (steps based directly on prior knowledge) and/or inductive steps (steps that go beyond prior knowledge). In addition, further instruction may be required to complete missing parts of an explanation, as described below.

A number of researchers have discussed learning from explanations involving inductive steps. These inductive steps may be based on naive theories of explanation or causality (e.g., TDL in OCCAM [Pazzani, 1991a], SWALE's explanation patterns [Schank and Leake, 1989]); or they may be based on unexplained features of the example(s) being explained (e.g., [VanLehn *et al.*, 1990; Rosenbloom and Aasman, 1990; VanLehn, 1987; Hall, 1986]). Taking inductive steps within explanations allows explanation-based approaches to do knowledge level learning [Dietterich, 1986], rather than simply compiling knowledge that is already present within the performance system. Knowledge level learning is clearly a requirement of learning from tutorial instruction, since the agent is expected to learn completely new tasks, unknown effects of operators, etc. (aspects of the requirement of learning of all types of knowledge), that cannot be learned by knowledge compilation alone.

Thus, the theory of learning from tutorial instruction is based on producing *explanations* of individual instructions as they apply to particular situations. The explanation process is *situated* in that the situation the instruction applies to provides the endpoints for explanation. Given an instruction, an agent:

1. **Determines the situation the instruction applies to.** This may be the current situation, or a hypothetical situation that must be constructed from the instruction by the agent.

2. **Attempts to explain how the instruction leads to goal achievement from the situation state.** If the agent can explain how performing the instruction in the situation leads to goal achievement (or to goal failure, for negative instructions), it can do general learning about the instruction. This is because the explanation indicates which features of the situation and instruction are crucial for success; those crucial features specify what situations the instruction should be applied to in the future. If an explanation cannot be formed, it indicates that the agent is missing some key piece(s) of knowledge – one of the "other steps" in Figure 2.2 is not known. At this point the agent may try to learn the missing knowledge needed to complete the explanation, or may abandon the explanation.

The following subsections elaborate on each of these.

### 2.2.2 Determining the situation an instruction applies to

A situation consists of an initial state and a goal to be achieved. In addition to a state and goal, situations also include a *goal expectation*; either success (successful goal completion), immediate success (goal completion is expected immediately after the current instruction is carried out), failure (goal completion blocked), or immediate failure (goal completion is expected to be blocked by carrying out the current instruction). If no situational features are specified in an instruction, it is implicitly situated, and applies to the agent's current situation. Alternatively, the language of the instruction can explicitly specify features of either the state or the goal the instruction is intended to apply to, making for two kinds of explicitly situated instructions. For example, "If the light is on, push the button" indicates a hypothetical state, that includes a light that is on. "To turn on the machine, flip the switch," indicates a hypothetical goal of turning on the machine.

Simple action commands ("Do X") are assumed to lead to success at achieving the goal; prohibited actions ("Never do X") indicate an expectation of failure to achieve the goal if X is performed; purpose clauses ("To do X, do Y") are assumed to indicate immediate success at achieving Y if X is performed.

As described in Chapter 4, the implemented agent presented here constructs hypothetical situations based on comprehending single instructions that specify either hypothetical states or hypothetical goals. Hypothetical situations that exist across multiple instructions are not supported. After the situation of application of an instruction has been determined, the agent attempts to explain how the instruction leads to the expected outcome in that situation.

### 2.2.3 Explaining an instruction

Explaining an instruction using only prior knowledge is preferable to making inductive leaps within the explanation. This is because presuming the prior knowledge is of high quality (correct and sufficiently general), the explanation and thus the learning will be of high quality also. Therefore, an instructable agent first tries to explain an instruction using prior knowledge.

The prior knowledge available to the agent is PSCM knowledge about operators and states. When executing (as opposed to explaining) actions, this knowledge applies to the current situation to select and apply operators and make inferences at appropriate points. The most natural way to use this knowledge when *explaining* an instruction is to apply it in the same way, but internally, to the situation associated with the instruction being explained. Thus, explanation takes the form of *forward internal projection*: the agent "imagines" itself in the instruction situation, and then runs forward, applying the step specified in the instruction, and the other knowledge that it has about states and operators, within the imagined situation. If the expected goal outcome is reached within this projection, then the projected path from the situation state through the instructed step to the goal comprises an explanation of the instruction. Forward projection is a common technique in planning (e.g., [Fikes and Nilsson, 1971; Hanks, 1990]; many others). In contrast to planning, however, the aim here is not to *search* by projecting possible paths, but to project the effects of a specified step.

Consider an instruction specifying an operator $OP$ to be applied in a state $S$, to achieve a goal $G$. The PSCM processing to internally project $OP$ is diagrammed in Figure 2.3. Processing

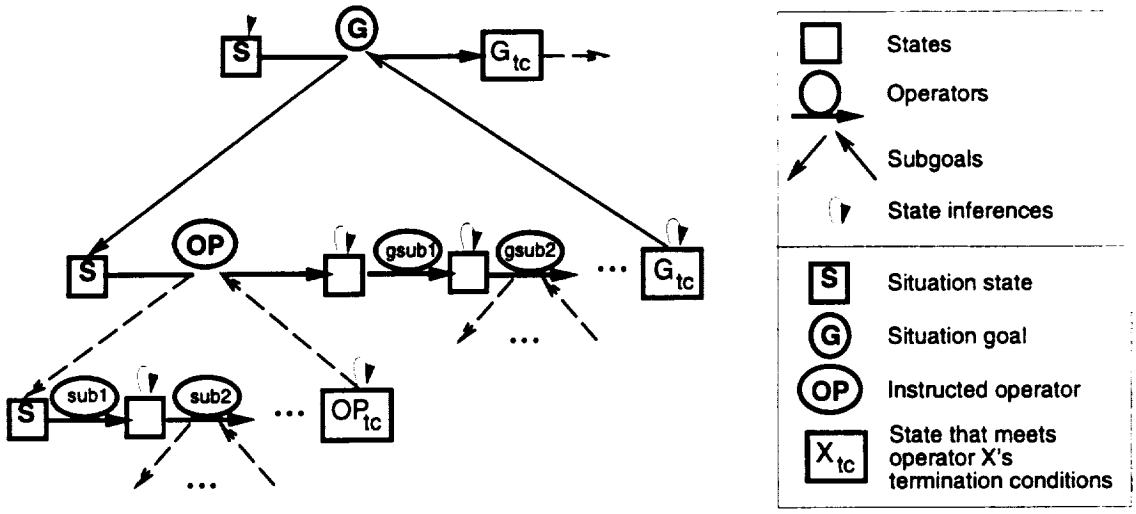



Figure 2.3: The structure of processing of the PSCM to explain (forward project) an instructed operator within a situation.

begins at the top left, in state $S$, by selecting operator $G$; arrows moving down to lower levels represent impasses that lead to processing in subgoals. The agent has placed itself (internally) into the situation of application of the instruction, with $S$ as the state and $G$ as a higher-level operator to be completed (recall that goals correspond to operators in the PSCM). Since $G$ cannot be completed directly, a subgoal arises. Within this context, the agent attempts to complete $G$ by simulating, to the best of its knowledge, performance of the instructed operator $OP$ (which may involve simulating sub-operators **sub1**, **sub2**, etc.). In essence, the agent has asked itself, "What will happen when I do what I was instructed to do?" After projecting $OP$, other knowledge applies within the projection as available, selecting and applying (simulating the effects of) operators (**gsub1**, **gsub2**,...), making inferences, etc. This is not a search, but direct application of prior knowledge to the given situation. The explanation of $OP$ is successful if the subgoal for achieving $G$ (that is, reaching the expected outcome) can be completed within the projection.

What knowledge must a PSCM-based agent possess to explain an instruction via internal projection? Consider again the explanation of an instruction commanding an operator $OP$ to be performed (Figure 2.3); instructions communicating other types of knowledge have similar explanation knowledge requirements. The explanation of an instruction involves knowledge about the operator specified by the instruction, $OP$, the goal of the situation, $G$ (an operator in PSCM terms), and the states the projection passes through.

Figure 2.4 elaborates the explanation by indicating the points where various types of knowledge about each of these components may apply during the projection process. The figure indicates that seven different categories of knowledge may be needed during an internal projection to explain an instruction:

(a). State inference knowledge may apply to any state in the internal projection.

(b). Mapping knowledge allows $OP$ to be produced and selected from the instruction.

(c). Knowledge of the effects of $OP$ (either directly or through a series of sub-operators applied in a subgoal) allows the state to be altered to reflect the (projected) execution of $OP$.

(d). If $OP$ is not a primitive operator, then projecting $OP$ requires the application of a series of sub-operators (**sub1**, **sub2**, ...) required to carry out $OP$. This involves operator proposal, operator preferential, and operator effects knowledge for the various sub-operators.

(e). Knowledge of $OP$'s termination conditions, or goal concept, allows the agent to terminate $OP$ at the appropriate point in the internal projection.

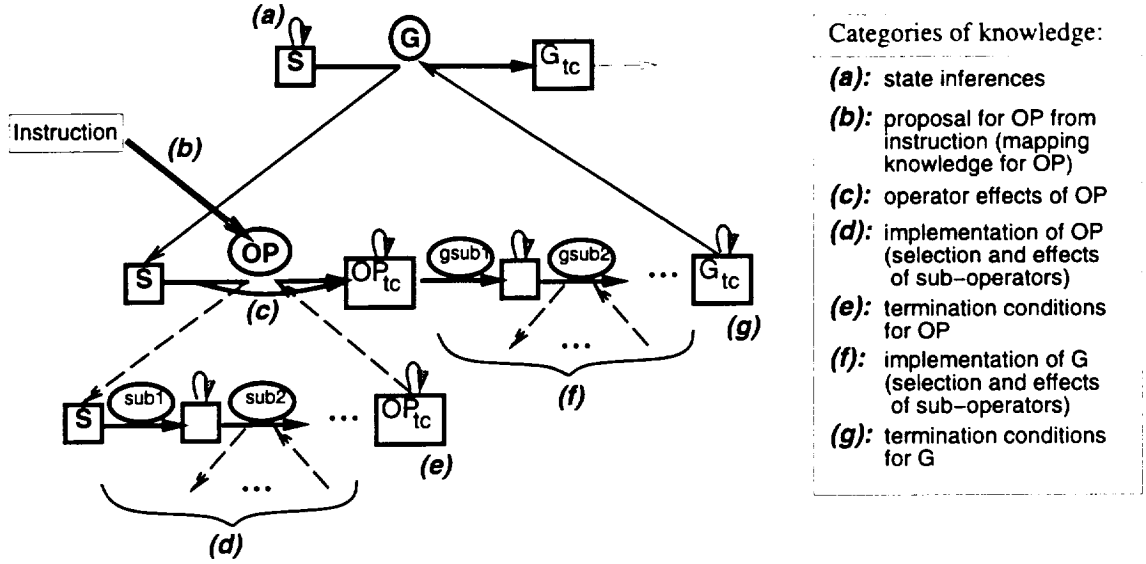



Figure 2.4: The types of knowledge that may be used in explaining an instruction.

(f). Once $OP$ has been projected, to complete the explanation the agent must know the rest of the implementation for $G$; that is, the selection and effects of a series of operators `gsub1`, `gsub2`,..., following $OP$ that allows $G$ to be completed. As in (d), this involves operator proposal, operator preferential, and operator effects knowledge for each sub-operator leading to completion of $G$.

(g). Finally, the agent must know the termination conditions of $G$, to recognize that the goal has been completed within the internal projection.

If all of the required knowledge in these categories (which include all of the five types of PSCM knowledge) is known to the agent, then the internal projection process will result in a successful explanation of $OP$ in the given situation. This explanation indicates the crucial features of $OP$, $S$, and $G$ required for success, allowing the agent to learn general knowledge about the use of $OP$ in states like $S$ to reach goals like $G$.

### 2.2.4 Dealing with incomplete explanations

The agent will be unable to complete an explanation if a needed piece of knowledge from any of these categories is unknown to the agent. An incomplete explanation is indicated by an unresolvable impasse during internal projection that precludes the completion of $G$. The possibilities of missing knowledge are summarized in Figure 2.5. For instance, the agent may not know a key effect of an operator, or a crucial state inference, making it unable to reach $G$'s termination conditions within its internal projection. Alternatively, a commanded operator $OP$ may be completely unknown and thus inexplainable.

More formally, consider an instructed operation $I$ that the agent must explain. A completed explanation would allow a piece of knowledge $I_K$ to be learned. If the agent cannot explain $I$ successfully, it is because another needed piece of knowledge $M_K$ (within one of the seven categories in Figure 2.5) is unknown. For example, $I$ might be a command to perform some operator $OP$; $I_K$, a proposal for $OP$; and $M_K$, knowledge of one of the effects of $OP$. Figure 2.6 shows an example of this kind (to be described in Chapter 4 below) for the instruction "To turn on the light, push the button." From this instruction, the agent should learn to propose pushing the button when it has the goal of turning on the light ($I_K$). It attempts to explain the instruction by simulating pushing the button (selecting the `push-button` operator in Figure 2.6) when the light is off. As the figure indicates, this simulation cannot be completed successfully if the agent is missing the knowledge of

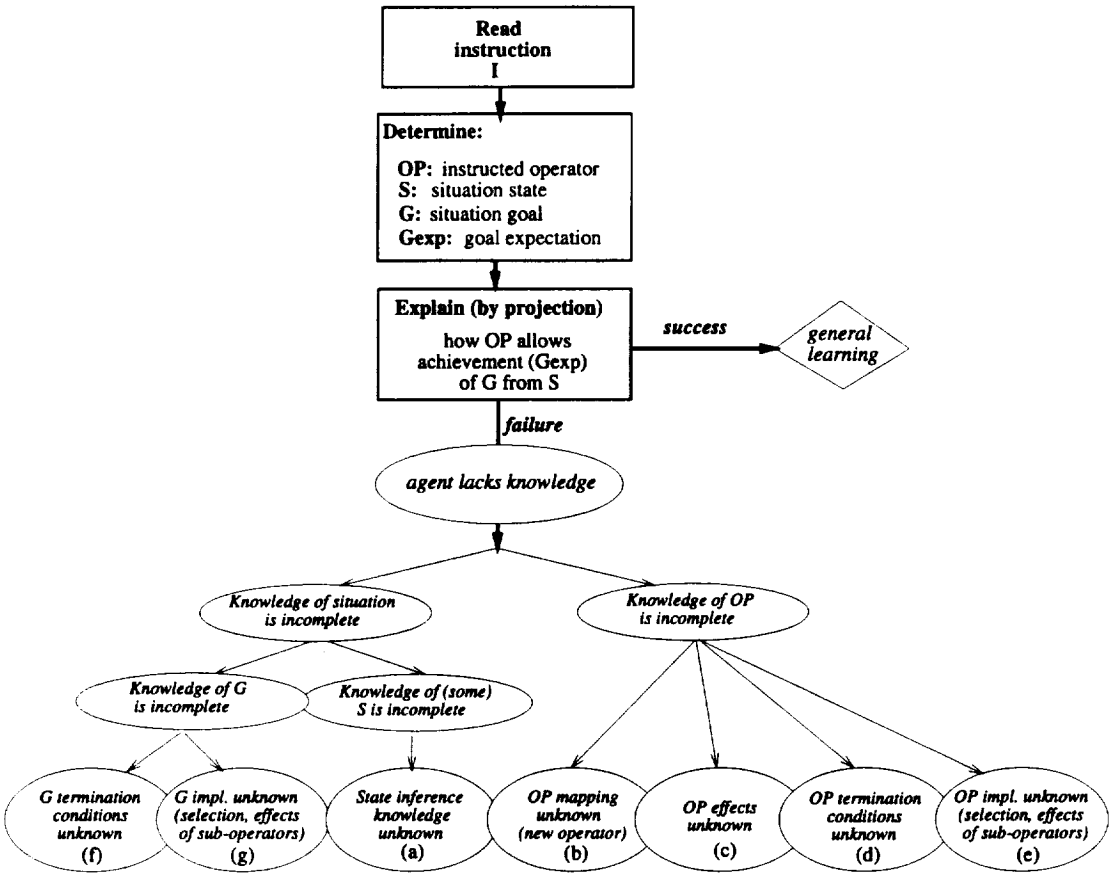



Figure 2.5: Types of missing knowledge that can cause an incomplete explanation.

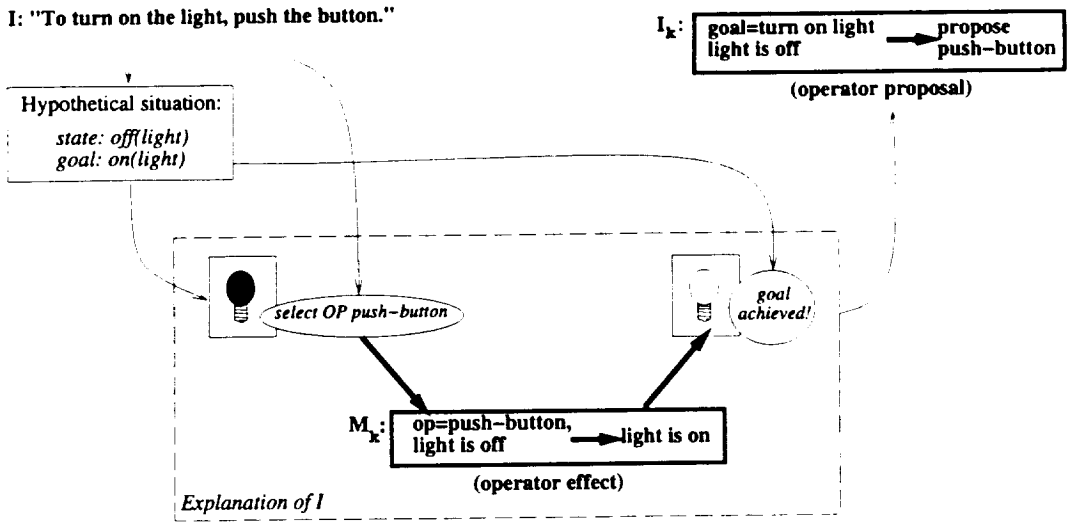



Figure 2.6: An example of missing knowledge needed to complete an explanation and allow learning from an instruction. $I$ is the instruction; $I_K$ is the knowledge learned from the instruction; and $M_K$ is the missing knowledge needed to complete the explanation of $I$.

how pushing the button affects the light ($M_K$). In general, of course, multiple pieces of knowledge needed for the explanation could be unknown (there could be multiple $M_K$'s).

Determining what knowledge $M_K$ is missing to cause an incomplete explanation is extremely difficult in general. It is very hard credit assignment problem for an agent to determine what knowledge it does *not* have, that if present would have led to a successful explanation. Thus, there is no algorithmic way to determine $M_K$. Rather, when the explanation of $I$ cannot be completed, the agent has the following possible courses of action:

1. **Give up on explaining $I$ now.**

   1.1. **Delay explanation until later.** In some cases, it is likely that the missing knowledge $M_K$ needed to complete the explanation will be acquired later. In such cases, the agent may delay completing the explanation of $I$ until the missing knowledge is learned. Initially, the agent can simply memorize what it was told to do (rote learning). Later, after the missing knowledge $M_K$ has been acquired, the previously memorized instruction, $I$, can be recalled and successfully re-explained to learn $I_K$.

   1.2. **Learn $I_K$ directly.** In some cases, the agent's knowledge is incomplete in ways that cannot be easily induced or learned later. In these cases, the agent can attempt to learn the knowledge to be gleaned from the instruction ($I_K$) directly, without an explanation to base the learning on. This might involve heuristics to induce $I_K$. In addition, the instructor might be asked to verify or alter $I_K$. For instance, if the agent cannot explain an instruction to perform $OP$, it might heuristically induce a set of conditions under which $OP$ should be selected, say $\{A, B, C\}$, and ask the instructor to edit this set as needed to produce $I_K$, a proposal for $OP$. This is the process followed by systems such as ASK [Gruber, 1989] and Robo-Soar [Laird *et al.*, 1989].

2. **Complete the explanation of $I$ by learning $M_K$.**

   2.1. **Induce $M_K$ to complete the explanation.** In some cases, the agent can make a good guess at what knowledge, $M_K$, is missing. In such cases heuristics can be used to induce the missing knowledge. Learning $M_K$ allows the explanation of $I$ to be completed, and $I_K$ to be learned.

| |
|---|
| 1. Give up on explaining $I$ now. |
| 1.1. Delay explanation until later. |
| 1.2. Learn $I_K$ directly (not from explanation). |
| 2. Complete the explanation of $I$ by learning $M_K$. |
| 2.1. Induce $M_K$ to complete the explanation. |
| 2.2. Take instruction to learn $M_K$. |

Table 2.3: Options when the explanation of an instruction $I$ cannot be completed because a piece of knowledge, $M_K$, is unknown.

| | | | |
|---|---|---|---|
| **1.1.** | if $M_K$ is likely to be learned from later instruction in the ongoing dialogue | $\longrightarrow$ | delay explanation until later |
| **1.2.** | if $M_K$ is not likely to be learned later, and no good guess about $M_K$ can be made | $\longrightarrow$ | learn $I_K$ directly (without completing the explanation of $I$) |
| **2.1.** | if a good heuristic guess at $M_K$ can be made | $\longrightarrow$ | induce $M_K$ to complete the explanation |
| **2.2.** | if $M_K$ is not likely to be learned later, and a heuristic guess at the type of $M_K$ can be made | $\longrightarrow$ | take instruction to learn $M_K$ |

Table 2.4: Conditions for following each option when the explanation of an instruction $I$ cannot be completed.

2.2. **Take instruction to learn the missing knowledge $M_K$.** Finally, the instructor can be asked to explain $I$ by teaching the agent $M_K$. This further instruction can allow $M_K$ to be learned, the explanation of $I$ to be completed, and $I_K$ to be learned. Similarly, inductive guesses at $M_K$ to complete the explanation (2.1, above) can be presented to the instructor for verification and alteration as needed.

This final option opens up the possibility of multiple levels of instruction. The agent can initiate instruction not only to perform tasks, but also to complete explanations. Rather than heuristically inducing $I_K$ when the explanation of $I$ cannot be completed (option 1.2), taking more instruction allows $I_K$ to be learned based on an explanation that includes $M_K$. Provided $M_K$ is a high quality piece of knowledge, $I_K$ will be of high quality as well. Thus, by composing knowledge together, multiple levels of instruction can lead to learning higher quality knowledge than a single level; learning at lower levels ($M_K$) supports learning at higher levels ($I_K$). In addition, $M_K$ is now available for use in future explanations. The examples given in Chapter 4 demonstrate this possibility.

For reference, the options for dealing with an incomplete explanation are listed in Table 2.3. The option to be followed depends on the properties of the missing knowledge $M_K$ that is involved. The key properties, as mentioned earlier, are whether $M_K$ is likely to be learned later in the ongoing dialogue with the instructor, and whether a good heuristic guess about $M_K$ can be made. The conditions for following each option are summarized in Table 2.4. These conditions are heuristics that are indicated by the type of situation and the point in processing where the explanation process breaks down. The options that Instructo-Soar follows under different circumstances will be described more specifically in the following chapters.

The theory does not include an explicit check for consistency when newly learned knowledge is added to the agent's knowledge base. However, as Kodratoff and Tecuci [1987] point out, learning apprentices are biased towards consistency to the extent that they only acquire new knowledge when current knowledge is insufficient, and that they use current knowledge when deriving new knowledge. In situated explanation, since the explanation process involves applying (projecting) the new knowledge in the appropriate situation, the agent maintains a kind of "performance consistency," rather than standard notions of semantic consistency. However, in some domains more explicit

measures of consistency (such as that used by ODYSSEUS [Wilkins, 1990]) may be required.

Even without explicit consistency checks, situated explanation meets the requirement that learning be incremental over the agent's lifetime, because it occurs as a part of the ongoing processing of the agent, and adds new pieces of knowledge to the agent's long term memory in a modular way. The least commitment, local control structure of the PSCM enables this property; new knowledge can be added to memory independent of current knowledge. If there is a conflict between pieces of knowledge, for example proposing two different operators to be performed in some situation, an impasse will arise that can be reasoned about or resolved with further instruction (again, maintaining performance consistency). An example of this is described in Chapter 4.

## 2.3 Comparison to alternative approaches

Situated explanation can be compared with theories of instructional learning that could be developed based on alternative learning approaches. Two major learning approaches will be considered: correlational learning (i.e., induction based on correlations of features) and case-based reasoning. When these approaches are constrained to meet the learning requirements of tutorial instruction, the resulting theories, although they differ in some ways, have many similarities to situated explanation. In particular, requirements 2 and 3 from Table 2.2, that learning occur quickly by making use of prior knowledge, and requirement 8, the use of both implicitly and explicitly situated instructions, strongly constrain the theories that could be developed.

### 2.3.1 Correlational learning

Correlational learning involves inducing concepts by correlating feature values that appear across multiple examples. Given enough examples, the correlations indicate which features are important to the concept being learned. Correlational techniques include symbolic concept learning and clustering methods (e.g., [Fisher, 1987; Quinlan, 1986; Michalski and Stepp, 1983; Mitchell, 1982]), connectionist methods (e.g., [Rumelhart and McClelland, 1986]), and other reinforcement learning techniques (e.g., [Sutton, 1988; Holland, 1986]).

Since instructions may be explicitly situated, determining the situation for an instruction would have to be included as a precursor to correlational learning, so that the features of the situation could serve as inputs to the learning algorithm.

In their pure form, correlational methods do not make use of prior knowledge; all knowledge is drawn from the examples themselves, with the aid of domain-independent biases. Because of this, the methods require a large number of examples for learning. An instructable agent, however, has prior knowledge that it is expected to use to produce fast, general learning. Thus, to be used in learning from instruction, correlational methods must be augmented with the ability to use prior knowledge to produce fast learning. A number of researchers have discussed ways to do this (e.g., [Thrun and Mitchell, 1993; Gordon and Subramanian, 1993; Pazzani, 1991b; Towell *et al.*, 1990; Flann and Dietterich, 1989]). This is the inverse to allowing inductive steps within explanation-based methods; both attempt to allow maximal use of prior knowledge without requiring that the prior knowledge is complete. If the prior knowledge that is available includes control and causal knowledge (operator selection and operator effects) – as it will for an instructable agent – then composing that knowledge to bias learning will (by definition) involve an explanation process.

Thus, once the requirements of instructability are applied, using correlational methods for learning from instruction appears to have many similarities to situated explanation. One difference that remains may be that while situated explanation always attempts to learn from a single example (making inductions and asking for verifications and explanations to do so), a theory with correlational learning as its basis could make use of multiple examples. However, induction over multiple examples could also be employed within an explanation-based approach (e.g., [Miller, 1993]).

### 2.3.2 Case-based learning

A second alternative would be to employ case-based reasoning methods (e.g., [Redmond, 1992; Alterman *et al.*, 1991; Schank and Leake, 1989]) to learn from instruction. A case might consist of the sequence of instructions for a procedure. When asked to perform the procedure again in some other situation, the previous case would be recalled and altered to apply to the current situation. This alteration process would result in learning of a more general form of the case.

Case-based learning has many similarities to explanation-based learning; the two can be thought of as points on a continuum. Two key dimensions of the continuum include:

- **Grain size of cases.** Case-based methods typically work with the full reasoning sequence used to reach a goal (e.g., a full sequence of instructions), while explanation-based methods often work with single steps in the reasoning (e.g., single instructions). As described earlier, in interactive instruction learning about individual steps appears to be more appropriate. However, the explanation process allows knowledge learned about each step in a sequence to affect learning of the remaining steps; the end result can be viewed as a distributed case or schema that encodes the overall procedure (as, e.g., [Alterman and Zito-Wolf, 1993]).

- **Generalization at storage versus retrieval time.** Case-based methods typically generalize cases primarily at retrieval time. That is, when a case is given, it is stored in a rote form. Later, the case is retrieved for use in some similar situation; based on altering the case for this situation, the case and its index are generalized. Explanation-based methods typically explain and generalize an example when it is given, and store the general knowledge derived from it. The theory presented here can be viewed as a blend of these approaches. The instructable agent attempts to explain an instruction when it is given. However, if the explanation cannot be completed, because of missing knowledge that will be learned later (option 1.1 from Table 2.4), the agent memorizes the instruction in a rote form, to be retrieved for explanation at a later time. This process is described in more detail in the next chapter.

As in correlational learning, a case-based instructable agent would still have to deal with explicitly situated instructions; here, the explicitly indicated features of the situation might be used as indices for retrieval of the case. Similarly, altering a case for use in a different situation would involve the use of prior knowledge, applied in an explanation type process. Thus, to meet the requirements of instructability, a case-based theory ends up with many similarities to situated explanation. As evidence of this, Alterman *et al.* [1991] describe a case-based approach to learning from written procedural instruction. Their system, FLOBN, uses instruction in altering an old procedure (using a pay phone) to produce a new, related procedure (using the Airphone). Although focused on altering a past cases, the process involves an explanation-based method for determining each instruction's role in the new procedure.

In summary, both correlational and case-based approaches, when constrained by the requirements of instructability, end up with many similarities to situated explanation. Whether the differences that remain would result in significant differences in performance of an tutorable agent is an empirical question.

## 2.4 Conclusion

This chapter has presented a theory of learning from tutorial instruction that meets many of the requirements for a complete tutorable agent described in Chapter 1. The first half of the chapter specified a computational model of an intelligent agent, the problem space computational model, and elucidated the types of knowledge within this computational model. The second half of the chapter described a learning technique, situated explanation, that allows an instructable agent to learn general knowledge from instruction.

In the next two chapters, the theory will be demonstrated by describing an implemented instructable agent, Instructo-Soar, that embodies the theory. Chapter 3 describes Instructo-Soar's learning of new procedures, and the extension of known procedures to new situations. Learning

procedures involves learning operator proposal and termination condition knowledge, from implicitly situated instructions. Chapter 4 describes learning of each of the remaining types of PSCM knowledge, and the handling of explicitly situated instructions.

# Chapter 3

# Instructo-Soar: Learning and Extending Procedures

This chapter and the following one present Instructo-Soar [Huffman and Laird, 1993b], an implemented instructable agent that embodies the theory of learning from tutorial instruction described in Chapter 2. Instructo-Soar is implemented in the Soar architecture [Laird *et al.*, 1987], and has been applied to a simulated robotic domain that includes a robot, tables, blocks, and various other objects to be controlled or manipulated. The Instructo-Soar agent engages in an interactive dialogue with its instructor, in which it receives instructions in the form of natural language sentences. From these instructions, it learns to perform tasks and extends its knowledge of the domain.

Instructo-Soar meets each of the transfer and interaction requirements for learning from tutorial instruction targeted in the previous chapters, as listed in Table 3.1. In particular, it is able to learn each of the types of knowledge it uses for task performance (the knowledge types of the PSCM) from instruction; it can learn from both implicitly and explicitly situated instructions; and it supports flexible interaction, allowing the instructor to request that a task be performed at any point, whether the agent knows about the task or how to perform it in the current situation. These capabilities, listed in Table 3.2, enable Instructo-Soar to exhibit a wider breadth of learning and interaction than previous learning apprentice systems.

Instructo-Soar is implemented in Soar productions. It begins with over 2,500 productions, or about 22,000 lines of code, and has learned up to 5,000 more productions. The simulated domain Instructo-Soar is applied to and its interface to Soar consist of about 10,000 lines of C code.

After a brief description of Soar and a statement of Instructo-Soar's domain and initial knowledge, this chapter discusses the learning of procedures from tutorial instructions that are implicitly situated (i.e., imperative commands). In particular, the chapter describes learning of completely new procedures, that the agent has no prior knowledge of, and learning to extend known procedures to apply in new, unfamiliar situations. In PSCM terms, a procedure corresponds to a complex operator, that must be selected, implemented, and terminated. Thus, learning a procedure involves learning selection knowledge for an operator, operator proposals for the series of sub-operators used to implement the operator, and the termination conditions for the operator. Chapter 4 completes the description of Instructo-Soar by describing its learning of the remaining types of PSCM knowledge, and its use of explicitly situated instructions. Appendix A details that agent's processing on an extended example that demonstrates its full range of learning and interaction capabilities.

## 3.1 Soar

Instructo-Soar is built within Soar, an architecture for problem solving and learning that follows the PSCM. Soar has been applied to a large number of problems in both AI and cognitive science (see [Rosenbloom *et al.*, 1993] for many examples), and has been proposed as the basis for a unified theory of cognition [Newell, 1990].

| | Requirement | From |
|---|---|---|
| 1. | General learning from specific cases | Transfer |
| 2. | Fast learning (each task instructed only once) | Transfer |
| 3. | Maximal use of prior knowledge | Transfer |
| 4. | Incremental learning | Transfer |
| 5. | Ability to learn all types of knowledge used in task performance | Transfer |
| 6. | Agent-initiated instruction | Interaction |
| 7. | Flexibility of knowledge content: full flexibility for action commands; others as appropriate | Interaction |
| 8. | Situation flexibility: both implicitly and explicitly situated instructions | Interaction |

Table 3.1: The transfer and interaction requirements of tutorial instruction to be met by the learning theory.

| Category | Type | | Described in |
|---|---|---|---|
| Knowledge types learned | operator proposal | | Chapter 3 |
| | operator termination | | Chapter 3 |
| | operator preferential | | Chapter 4 |
| | operator effects | | Chapter 4 |
| | state inference | | Chapter 4 |
| Situation flexibility handled | implicitly situated | | Chapter 3 |
| | explicitly situated: | hypothetical state | Chapter 4 |
| | | hypothetical goal | Chapter 4 |
| Flexibility of content | full flexibility for action commands | | Chapter 3 |
| | others as appropriate | | Chapter 4 |

Table 3.2: Instructo-Soar's coverage of knowledge and interaction types.

In Soar, as in the PSCM, all activity occurs by applying operators to states within problem spaces. When a Soar agent cannot make progress within a problem space, an impasse arises, and a subgoal is generated to resolve the impasse.

The major component of Soar that has not been discussed in the description of the PSCM is learning. Learning occurs in Soar whenever results are returned from a subgoal. The learning process, called chunking, produces new rules, or *chunks*, that summarize the processing in the subgoal that led to the creation of the result. Depending on the type of result, chunks may correspond to any of the types of PSCM knowledge shown in Table 2.1. When similar situations arise in the future, chunks allow the impasse that caused the original subgoal to be avoided by producing their results directly. Chunking is a form of explanation-based learning [Rosenbloom and Laird, 1986]. Although it is a summarization mechanism, through taking both inductive and deductive steps in subgoals, chunking can lead to both inductive and deductive learning. Chunking occurs continuously, making learning a part of the ongoing activity of the agent.

## 3.2 The domain

The primary domain Instructo-Soar has been applied to is the simulated robotic domain shown in Figure 3.1. A 3-D graphical simulator for the domain has been built that runs on Silicon Graphics machines. Instructo-Soar's techniques have also been applied in a more limited way to a flight domain [Pearson *et al.*, 1993], in which Soar controls a flight simulator and instructions are given for taking off

The agent in Figure 3.1 is a Hero robot with one arm and gripper. The robot is in a room that contains tables, blocks of different sizes and made of different materials, an electromagnet, and a light that is controlled by two push buttons. The gripper can grasp small blocks and the magnet,

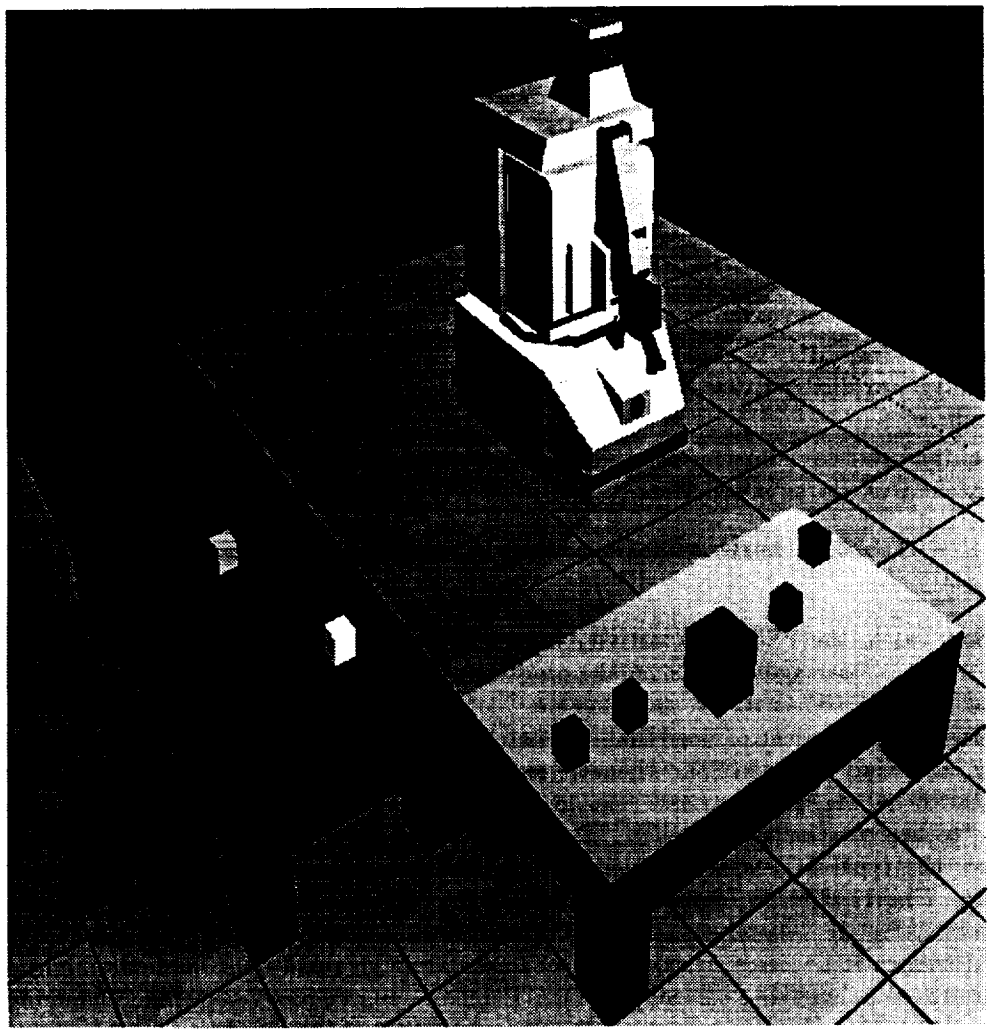


Figure 3.1: The robotic domain Instructo-Soar has been applied to.

Properties of single objects:
type (block, magnet, etc.)
location
size
color
gripper status (open/closed)

Relations between objects:

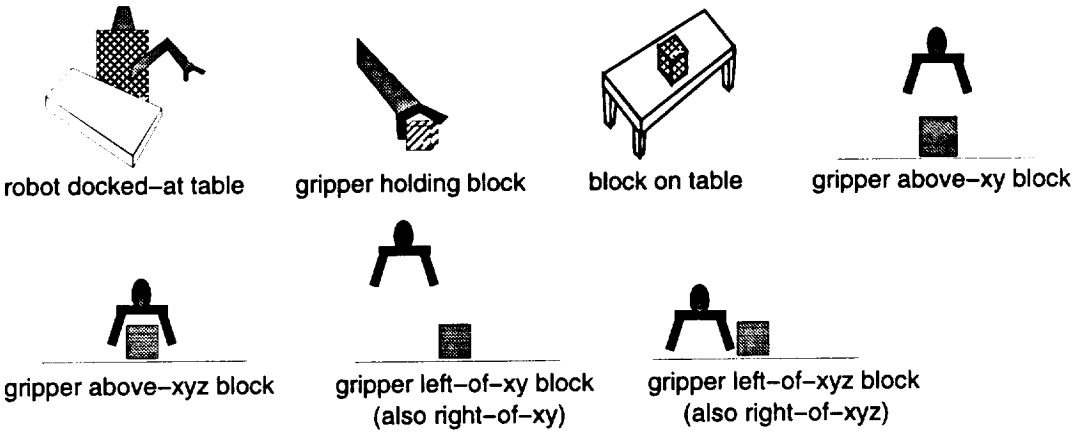



Figure 3.2: The basic properties and relationships known to Instructo-Soar in its robotic domain.

but does not fit around large blocks. Blocks made of metal (either small or large) can be picked up using the magnet. The magnet's status (on or off) is toggled whenever it is squeezed; that is, whenever the gripper is closed around it. When off, the magnet is red; when on, it is white. The light may be off, on and dim, or on and bright. A red button toggles the light on or off; a green button toggles it dim or bright, when it is on.

## 3.3 The agent's initial knowledge

Instructo-Soar is made up of a set of problem spaces within Soar. The problem spaces implement an agent with three main categories of knowledge: natural language processing knowledge, originally developed for NL-Soar [Lewis, 1993; 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. The agent's knowledge of how to obtain, use, and learn from instructions codifies the theory of learning from tutorial instruction.

The Instructo-Soar agent begins with the ability to interact with its instructor and to read instructions it is given. The agent does not expand its interaction or natural language capabilities *per se* as it takes instruction, although it does learn how sentences map onto new operators that are learned. Reading is done by an augmented version of NL-Soar [Lewis, 1993; Lehman *et al.*, 1991], a natural language theory being developed within Soar. NL-Soar learns to speed up its processing as it reads more and more sentences.

The Instructo-Soar agent has the ability to perceive the world around it, and to recognize a set of basic object properties and relationships between objects. These properties and relationships are shown in Figure 3.2. The agent perceives the entire visible world at once; shifting of perceptual attention is not currently modeled.

The agent's initial task knowledge is listed in Table 3.3. It includes the knowledge of a set of primitive operators: moving to tables, opening and closing the hand, and moving the arm up, down, and into relationships with objects (e.g., above, left of, or right of things). The agent knows how natural language sentences map on to each of these operators, and how to execute each operator externally. It also has an incomplete knowledge of the effects of these operators, that is used when

Initially known:

- knowledge to map from NL, execute, and simulate primitive operators
  - **move-to-table**
  - **move-arm(up/down)**
  - **move-arm(above, left-of, right-of <object>)**
  - **open/close-gripper**
  - **switch-status(light or magnet, on or off)** *(no execution knowledge)*
- knowledge about holding objects
  - closing the hand around a small object causes it to be held
  - moving the hand when holding causes the held object to move

Initially unknown (all learned in the example of Appendix A):

- no knowledge of complex operators
- no knowledge about controlling the light or magnet
  - how pushing the buttons affects the light
  - how grasping the magnet affects its status
- no ability to recognize the magnet's status
- no knowledge about magnetism – the magnet's effect on metal objects
- no ability to recognize the magnet **stuck-to** other objects
- no knowledge of actions to avoid (e.g., grasping explosive objects)

Table 3.3: The initial domain knowledge of Instructo-Soar.

they are internally projected. However, some effects of these operators are unknown to the agent.

The agent knows about holding things; it knows that if it closes its hand around a small object, then that object is now being held; and it knows that it cannot close its hand around a large object. However, the agent has very little knowledge about the magnet and the light. It understands what it means to turn something on or off, but does not know how to turn the magnet or light on or off or to change the light's brightness. In addition, the agent does not know anything about magnetism; i.e., that the magnet will attract metal objects. Finally, that agent knows nothing about complex operators that involve combinations of the primitive operators; e.g., picking up or lining up blocks.

Thus, the agent's initial domain theory is incomplete in a number of different ways. By learning from instruction, it will fill in and expand its knowledge of the domain. Instructo-Soar does not deal with *incorrect* domain knowledge; the knowledge it begins with is assumed to be correct, although incomplete. This is an area for future work, as described in Chapter 6.

The following sections describe the learning of a new procedure, the extension of a known procedure to apply in a new situation, and procedure learning when faced with incomplete underlying domain knowledge. More technical details about Instructo-Soar's implementation can be found in [Huffman, 1992]. In what follows, the instructor's inputs are given in **typewriter font** and Instructo-Soar's responses in *italics*.

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.

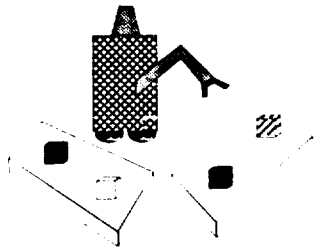


Figure 3.3: Instructions given to Instructo-Soar to teach it to pick up a block.

## 3.4 Learning a new procedure

This section describes Instructo-Soar's ability to learn completely new procedures from instruction. To explain Instructo-Soar's performance, the example of teaching the agent to pick up blocks will be used. One possible set of instructions for this task is shown in Figure 3.3.

A procedure takes the form of an operator in the PSCM (and thus in Soar). Since picking up things is not a known procedure initially, when told "Pick up the red block," the agent must learn a new operator.

To be executed in response to a natural language command, and operator must be selected, implemented, and terminated. To select an operator based on a natural language command, the agent must know the argument structure of the operator (a template), and how natural language maps onto that argument structure. Thus, learning a new operator involves learning each of the following:

1. **Operator template:** The agent learns what the operator's arguments are 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 object that isn't currently picked up.

2. **Mapping from natural language:** The agent learns to map from natural language semantic structures to an instantiation of the new operator, so that the operator can be selected when commanded in the future. For picking up blocks, the agent learns to map the semantic object of "Pick up ..." to the single argument of its new operator template.

3. **Implementation:** The agent learns how to perform the operator. New operators are performed by executing a sequence of primitive and/or previously learned operators, so the implementation takes the form of selection knowledge for these sub-operators (e.g., moving to the proper table, moving the arm, etc.)

4. **Termination conditions:** The agent learns to recognize 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.

The operator template and the natural language mapping are involved in the mapping problem. Instructo-Soar uses a fairly straightforward approach to the mapping problem, and thus to learning these parts of a new operator. Namely, the agent assumes that the argument structure of the natural language command used to request the unknown operator is the required argument structure of the operator itself. It learns a template containing the arguments in the argument structure (e.g., `object`), and a mapping from the argument structure to the new operator template. This approach is sufficient for imperative sentences with direct arguments; it would fail for commands with complex arguments, such as path constraints ("Move the TV into the other room, keeping it as far from the heater as possible").

To learn a general operator implementation from instructions, an agent must solve the transfer problem; that is, it must determine the proper scope of applicability of each instruction used in performance of the new operator. Consider the instruction "Move to the yellow table." Under what

conditions should the agent propose moving to a table, and what features determine which table? Instructo-Soar uses the situated explanation technique described in the previous chapter to solve this problem; it attempts to explain how each instruction leads to successful completion of the new operator. 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, making it impossible to explain how any of the instructions lead to achievement of those conditions, as diagrammed in Figure 3.4.

The agent will learn the termination conditions of the new operator when its initial execution is completed. Thus, during the initial execution, the agent gives up on explaining the instructions for the new operator, and falls back on simply memorizing the instructions as they are given (option 1.1 from Table 2.3). This initial learning is based on a weak inductive step: believing what the instructor says. The learning is rote and overspecific, with an "episodic" flavor. At the end of the initial execution, the termination conditions of the new operator can be induced. After this, the agent *can* form explanations of how each instruction, recalled from its episodic memory, allows the goal (the termination conditions) to be reached, using a forward internal projection. Based on these explanations, a general implementation sequence for the new operator is learned.

The details of learning a new procedure will be illustrated with the example of teaching the agent to pick up blocks. The agent is given the instructions in Figure 3.3 during the course of performing the task.

### 3.4.1 First execution

After the first execution of a new procedure, the agent must be able to perform at least the exact same task without being re-instructed, as specified by requirement 2 ("fast learning") of the requirements for learning from tutorial instruction (Table 3.1). Thus, the agent must learn, in some form, each of the parts of the new operator described earlier during the first execution. If some part of the operator were left unlearned, the agent would not be able to repeat the task without more instruction.

The first instruction is "Pick up the red block." The language is comprehended to produce a semantic structure and resolve "the red block" to a block in the agent's environment. However, the semantic structure produced does not correspond to any known operator, indicating to the agent that it must learn a new operator. Thus, a new operator template is generated (e.g., `new-op14`), with an argument structure that directly corresponds to the semantic structure's arguments (here, one argument, `object`). The agent learns a mapping from the semantic structure to the new operator, so that if presented with a similar natural language request in the future, it will map to the new operator.

Next, the new operator is selected for execution. Since the agent doesn't know any implementation for the operator, it immediately reaches an impasse and asks for further instructions. Each instruction in Table 3.3 is given, comprehended and executed in turn. These instructions provide the implementation for the new operator. They are implicitly situated; that is, each instruction applies to the current situation that the agent finds itself in.

Ultimately, the implementation sequence will be learned at the proper level of generality, based on explaining how each instructed operator leads to successful execution of the new operator. During the initial execution, however, this is impossible, because the goal of the new operator is not yet known. As shown in Figure 3.4, the unknown goal precludes completing explanations. In addition, the agent does not know the "other steps" (further instructions) needed after the current instruction.

Thus, the missing knowledge needed to complete the explanation of each instruction includes knowledge of the steps following the instruction and the termination conditions of the new operator. Both of these will be available once the performance of the new operator has been completed. Thus, the agent follows option 1.1 from the set of options for dealing with an incomplete explanation given in Table 2.4; namely, it gives up on explaining each instruction until after the missing knowledge is learned. However, the agent is required to learn each implementation step in some form, so that it can perform at least this same task in the future without requiring further instruction. Thus, the

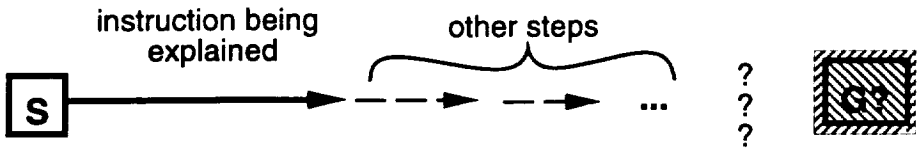



Figure 3.4: Instructions teaching a new operator cannot be explained before the goal concept, or termination conditions, of the new operator are learned.

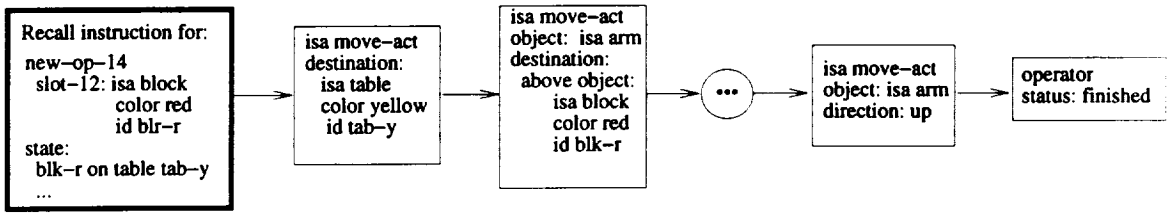



Figure 3.5: The episodic case learned as a side effect of comprehending the instructions for a new procedure.

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. These rules allow NL-Soar to resolve referents in later sentences, by providing a memory of past referents (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. The episodic case that results corresponds to a lock-step, overspecific sequencing of the instructions given to perform the new operator, as illustrated in Figure 3.5. For instance, the agent encodes that "to pick-up (that is, new-op14) the red block, I was first told to move to the yellow table." Although drawn as a single structure in Figure 3.5, this encoding actually exists across multiple rules, each specifying a single semantic feature of each instruction. The properties of this episodic memory are discussed in more detail in Chapter 5.

One issue that arises here is whether and when to generalize the index and the information contained within the episodic case. However, at this point any generalization would be purely heuristic, since the agent is unable to explain the various steps of the episode. Thus, Instructo-Soar takes the overly conservative approach of leaving the case in a fully specific, rote form. There is some evidence that people are also conservative in generalizing memory of procedural steps [Lewis, 1988].

Finally, after carrying out all of the steps for picking up the red block, the agent is told "The operator is finished," indicating that the goal of the new operator has been achieved. An explicit indication is required because the agent has no concept of "pick up" before instruction. The indication that the new operator's goal has been met triggers the agent to learn the termination conditions for it.

Learning termination conditions is an inductive concept formation problem. The termination conditions, or goal concept, of a new operator being learned cannot be derived from the agent's current knowledge, since the correct concept is chosen completely by the instructor. Standard concept learning approaches may be used here; however, typically, an instructor will expect learning within a small number of examples. Instructo-Soar learns the termination conditions for a new operator from a single example. It heuristically induces a set of termination conditions that are a subset of the current state, and then allows the instructor to alter, and finally verify, this set of conditions.

The heuristic used to induce an initial guess at a new operator's termination conditions is simple: the current (final) state of the world is compared to the initial state the agent was in when

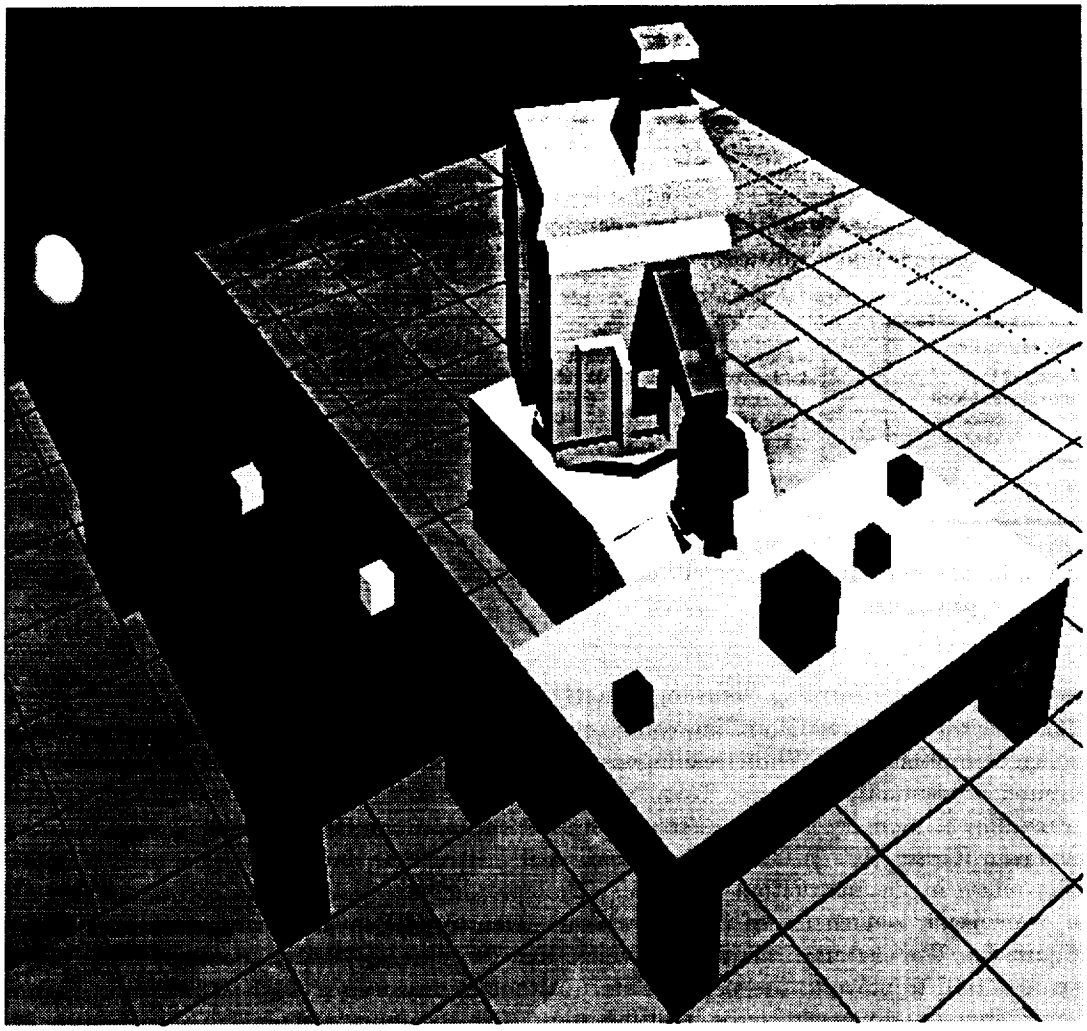


Figure 3.6: The final state after picking up the red block.

commanded to perform the new operator. Everything that has *changed* is considered to be a part of the termination conditions of the new operator. The final state in this case is shown in Figure 3.6. The changes from the initial state (Figure 3.1) are that the robot is standing at a table, holding a block, and the block and gripper are both up in the air. The colors and object types of the block and table are not included because they did not change.

This simple heuristic forms the system's inductive bias for learning termination conditions. It gives a reasonable guess, 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.

Thus, after making the initial inductive guess, the agent presents the induced set of termination conditions to the instructor, to be altered and eventually verified. The interaction that occurs for "Pick up the red block" is shown in Table 3.4. The instructor indicates that the robot's being docked at a table is not important to the goal of picking up an object. The instructor can add features to the goal concept in a similar way (an examples of this is given in Appendix A). Finally, the instructor verifies the goal concept, and Instructo-Soar learns the set of termination conditions for the new "pick up" operator.

Instructo-Soar performs the inductive learning of termination conditions by EBL (chunking) over an overgeneral theory – one that is able to generate any of the features of the current state

*So I guess doing "pick up the block" means:*
*the robot is docked at the table*
*the gripper is closed*
*the gripper is holding the block*
*the gripper is raised*
*the block is raised*
*Is that right?*
`> The robot need-not-be docked-at the table.`
*So I guess doing "pick up the block" means:*
*the gripper is closed*
*the gripper is holding the block*
*the gripper is raised*
*the block is raised*
*Is that right?*
`> Right.`

Table 3.4: Instructo-Soar's interaction with its instructor to learn the termination conditions of a new operator. The dashes between words in the instructor's input simplify natural language comprehension, which is not the focus of this thesis.

(similar to [Miller, 1993; VanLehn *et al.*, 1990; Rosenbloom and Aasman, 1990]). This type of inductive learning has the advantage that the agent can alter its inductive bias to reflect other available knowledge as appropriate. In this case, the agent uses knowledge gleaned from further instruction (the instructor's indications of features to add or remove) to alter the induction. Other knowledge sources that could be employed include analogy to other known operators (e.g., pick up actions in other domains), domain specific heuristics, etc.

The process of learning termination conditions can be viewed as forming an explanation of the instruction "The operator is finished." The agent cannot explain why the operator (e.g., "pick up") is finished because it does not know the termination conditions of the operator. The termination conditions are the missing knowledge ($M_K$) needed to complete the explanation of the instruction. In this case the agent can make a good guess at $M_K$ using its heuristics for inducing termination conditions (option 2.1 of the possible options for dealing with incomplete explanations listed in Table 2.4). The agent applies its heuristics and then moves to option 2.2 (taking further instruction), allowing the instructor to alter and verify the induction of $M_K$.

Through the first execution of a new operator, then, the agent:

- Carries out a sequence of instructions achieving a new operator.
- Learns an operator template for the new operator.
- Learns the mapping from natural language to the new operator.
- Learns a rote execution sequence for the new operator, as a side effect of natural language comprehension.
- Learns 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 a rote, episodic memory, it can only perform the *exact* same task. It has not learned generally how to pick up blocks yet.

**If** the goal (higher-level operator) is **new-op-14** with **object <obj>**, and
**<obj>** is sitting on a table **<table>**, and
the robot is not docked at **<table>**, and
**<obj>** is **small**, and
the gripper has **status open**,
**then** propose operator **move-to-table(<table>)**.

Figure 3.7: The general operator proposal rule learned from the instruction "Move to the yellow table" (**new-op-14** is the newly learned "pick up" operator).

### 3.4.2 Generalizing the new operator's implementation

The agent now knows the goal concept and full (though rote) implementation sequence for the new operator. Thus, it now has the knowledge that it needs to explain how each instruction in the implementation sequence leads to goal achievement, provided its underlying domain knowledge is sufficient. By explaining each instruction in the sequence, the agent will learn the implementation of the new operator in a general form that will transfer to an appropriate range of similar situations.

Each instruction is explained by recalling it from episodic memory and internally projecting its effects and the rest of the path to achievement of the termination conditions of the new operator. The explanation indicates the crucial features of the instruction and situation that lead to success. These crucial features are combined into a general rule that proposes the instructed operator under the right conditions. For example, the rule learned for the instruction, "Move to the yellow table" is shown in Figure 3.7. The rule generalizes the original instruction by not including the color of the table, because it was not critical for achievement of the "pick up" goal. It specializes the original instruction by adding the fact that the table has the object to be picked up sitting on it, and that the object is small (only small objects can be grasped by the gripper). The rule also includes the fact that the gripper is open, because this was important to being able to grasp the block in the instructed case. The details of how Soar's chunking mechanism produces such rules based on forward projections are described in Appendix C.

After learning the general proposal rules for each step in the instruction sequence, the agent will perform the task without reference to the rote case. For instance, if asked to "Pick up the green block," the rules for mapping language semantics to an operator apply, causing **new-op14** to be generated and instantiated with the green block as its argument. **New-op14(green block)** is selected to be performed. Then, the general sub-operator proposal rules for **new-op14** fire one by one as they match the current situation, to implement the operator. For instance, when the robot is not docked at the proper table, the rule shown in Figure 3.7 will match, causing the **move-to-table** operator to be selected and performed. After the agent has performed all of the implementation steps of **new-op14**, it recognizes that the termination conditions are met (the gripper is raised and holding the green block), and **new-op14** is terminated.

Since the general proposal rules for implementing the task are directly conditional on the state, 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 that was initially learned applies only when starting from the original initial state, and it is not reactive because its steps are not conditional on the state.

Instructo-Soar learns general rules from instructions for a new procedure by recalling and explaining them at some point *after* their first execution. An alternative would be to try to explain the instructions *during* their first execution. During-execution explanation would require observing the effects of each instructed action when it is executed and explaining (deriving from knowledge) which state properties lead to each observed effect (for example, closing your hand when *above* a block causes it to be held, while other features, such as the block's color, are not causally related). Thus, the agent would have to keep track of not only the observed state, but also the derivations of each observation – essentially, a second state representation. At the end of the first execution, the

important net effects of the instruction sequence are determined, in the form of learning the new operator's termination conditions. After learning these termination conditions, an agent forming explanations during the first execution would chain backwards from the termination conditions through its explanations of the effects of each instruction, to build an overall explanation of all of the instructions. This kind of processing is a viable alternative to the post-execution recall and explanation process used here. However, it has the disadvantage that the agent must keep track of both an observed and a derived state, and keep the two in correspondence.

### 3.4.3 Recall strategies

The agent has a number of options for *when* to recall and form explanations of the instruction sequence. The possibilities include:

1. **Introspective recall.** The agent recalls and attempts to explain the full sequence of instructions for the new operator immediately after completing the new operator and learning its termination conditions.

2. **Multiple recall.** The agent recalls and attempts to explain the full sequence of instructions for the new operator upon being asked to perform the operator again starting from the same initial state.

3. **Single recall.** The agent recalls and attempts to explain single instructions in the sequence upon being asked to perform the operator again starting from the same initial state. That is, at each point in the execution of the operator, the agent recalls the next instruction, and attempts to explain it by forward projecting it; but if this projection does not result in a path to goal achievement, rather than recalling the next instruction in the sequence to continue the forward projection, the agent gives up on explaining this instruction and simply executes it in the external world.

Each of these recall strategies is implemented in Instructo-Soar. Which strategy to use is a parameter of the agent; the agent cannot dynamically select among strategies while it is running. In what follows, each recall strategy is described in more detail.

#### Introspective recall

In the introspective recall strategy, the agent recalls and explains ("introspects about") the entire instruction sequence immediately after completing its first execution of the new operator being learned. This involves an internal forward projection of multiple operators, beginning at the state the agent was in when the new operator was first suggested. In the "pick up" case, it is the state in which the robot is standing in the corner of the room, with its arm down and holding nothing. Starting (internally) from this state, the agent recalls and simulates each of the instructions in turn, eventually reaching the termination conditions of the new operator. This multiple operator projection explains each of the operators in the implementation sequence, provided the termination conditions of the new operator are achieved.

The advantage of this strategy is that the agent learns a fully general version of the new operator right after its first execution. The agent can pick up other objects right away.

The strategy has three disadvantages. First, it requires that the agent remember the initial state in which it was commanded to perform the new operator. This may be difficult if the amount of information in the state is large (although it is not in the robotic domain being used here). When there is embedded instruction for multiple new operators, multiple states must be remembered.

Second, the multiple step recall and simulation required (forward projection of the entire sequence of instructed operators) can be time consuming. The introspection requires an amount of time proportional to the length of the sequence of instructions. Thus, the agent's ongoing performance of the tasks at hand is temporarily suspended. This could be awkward if the new procedure being learned is nested within a larger task that must still be completed, or if the agent is under pressure to act quickly.

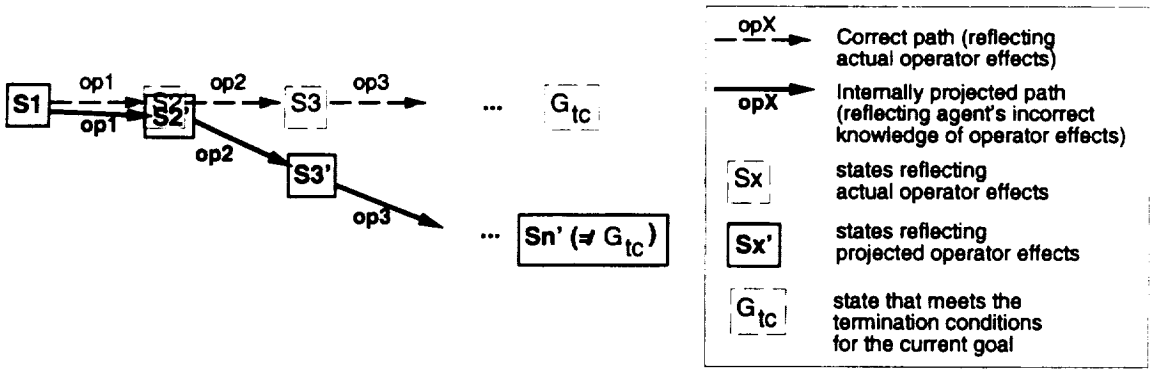



Figure 3.8: The multiple step projection required by the introspective and multiple recall strategies can result in incomplete explanations due to compounding of errors in the agent's domain knowledge.

Third, multiple step projection is susceptible to the compounding of errors in the agent's underlying domain knowledge. The projection of each successive operator begins from a state that reflects the agent's knowledge of the effects of the prior operators in the sequence, as shown in Figure 3.8. If this knowledge is incomplete or incorrect, then the state can move further and further from reflecting the actual effects of the prior operators. This can lead to incomplete explanations, or (more rarely) to spuriously successful explanations (e.g., reaching success too early in the instruction sequence). Minor domain knowledge problems in the knowledge of individual operators, that alone would not result in an error in a single step explanation, may combine within the projection to cause an error.

The single recall strategy overcomes the problems of multiple step projection, but requires multiple executions to learn the full general implementation of the new operator, as described below.

### Multiple recall

In the multiple recall strategy, the agent waits to recall and explain the full implementation sequence of the new operator until asked to perform the new operator a second time, starting from the same initial state. When asked to again pick up the red block, the agent selects the newly learned operator, and then recalls the instructions it was given during the first execution and attempts to explain them. As in the introspective strategy, in the multiple recall strategy the agent recalls and projects multiple operators. However, this strategy does not require that the agent recall the initial state for the projection. For small domains like Instructo-Soar's robotic domain, this is only a minor advantage. The disadvantage of this strategy as compared to the introspective strategy is that it requires two executions of the new operator to learn a general implementation.

### Single recall

Like the multiple recall strategy, in the single recall strategy the agent waits until asked to perform the new operator again before attempting to recall and explain the implementation instructions. In this strategy, however, the agent only recalls a single instruction to internally project at a time. After the recalled operator is projected, the agent applies whatever general knowledge it has about the rest of the implementation of the new operator. If the agent does not know the rest of the generalized path to complete the new operator, however, it does not recall any further instructions in the sequence. Rather, the internal projection is terminated and the single recalled operator is applied in the external world.

The single recall strategy requires a number of executions of the new operator equal to the length of the instruction sequence to learn the general implementation. This is because limiting recall to a

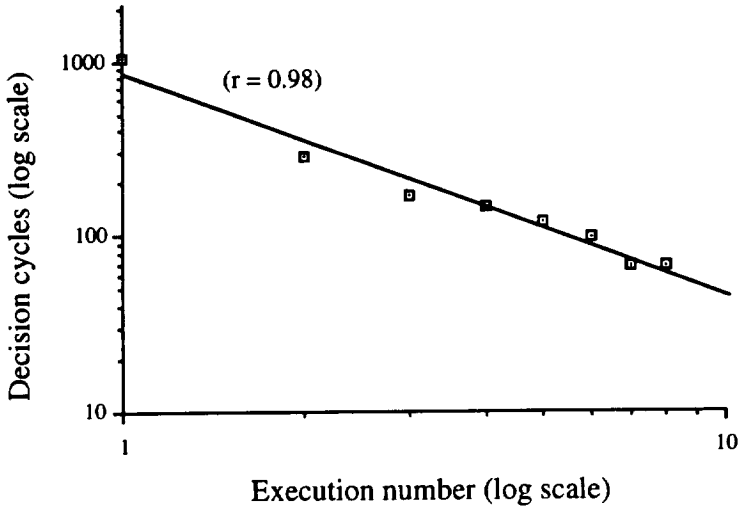



Figure 3.9: Decision cycles versus execution number to learn "pick up" using the single recall strategy.

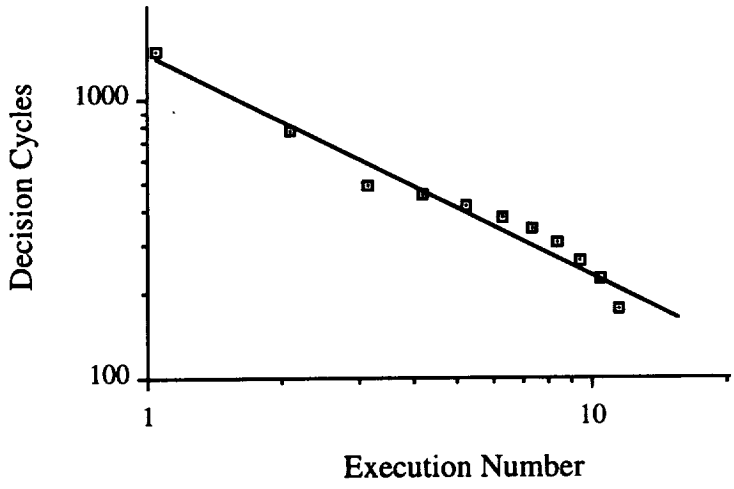



Figure 3.10: Decision cycles vs. execution number to learn to move objects left of one another using the single recall strategy.

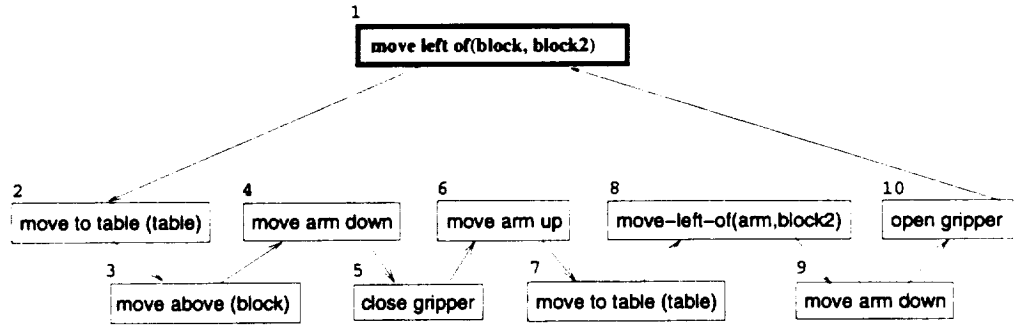



Figure 3.11: A graphical view of a flat instruction sequence for `move-left-of(blk1,blk2)`.

single step allows only a single sub-operator per execution to be generalized. This is single recall's major disadvantage as compared to the other strategies, but also leads to some interesting learning characteristics:

- **Bottom-up learning**. Generalized learning has a "bottom-up" form, starting at the end of the implementation sequence and moving towards the beginning. On the second execution of the new operator, a path to the goal is known only for the last instruction in the sequence (it leads directly to goal completion), so a general proposal for that instruction is learned. On the third execution, after the second to last instruction is internally projected, the proposal learned previously for the last operator applies, leading to goal achievement within the projection and allowing a general proposal for the second to last instruction to be learned. This pattern continues back through the entire sequence until the full implementation has been learned generally. As Figure 3.9 shows for learning "pick up", the resulting learning curve closely approximates the power law of practice [Rosenbloom and Newell, 1986] ($r = 0.98$). Figure 3.10 shows a similar curve for learning to move objects left of one another ($r = 0.98$).

- **Effectiveness of hierarchical instruction.** Due to the bottom-up effect, the agent learns a new procedure more quickly when its steps are taught using a hierarchical organizations than when they are given in a flat sequence. Figure 3.11 depicts a flat, nine step instruction sequence for teaching Instructo-Soar to move one block left of another; Figure 3.12 depicts a hierarchical instruction sequence for the same procedure, that contains 13 instructed operators, but a maximum of three in any subsequence. By breaking the instruction sequence into shorter subsequences, a hierarchical organization allows multiple subtrees of the hierarchy to be generalized during each execution. 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 $2\sqrt{N}$ executions are required for full general learning. The number is $2\sqrt{N}$ rather than only $\sqrt{N}$ because lower subtrees in the hierarchy must be generalized before they can be integrated into higher subtrees. For instance, for the hierarchy of instructions in Figure 3.12, **grasp**'s implementation must be generalized before **grasp** can be included in the general implementation for **pick up**. More broadly, a sequence of $N$ actions taught as an $H$-level hierarchy with $\sqrt[H]{N}$ subtasks in each subsequence takes $H \cdot \sqrt[H]{N}$ executions to generalize. The hierarchy in Figure 3.12 has an irregular structure, but still results in a speedup because the length of every subsequence is small (in this case, smaller than $\sqrt{N}$). As shown in Figure 3.13, the flat sequence of Figure 3.11 takes nine ($N$) executions (after the initial execution where the instructions are given) to generalize, whereas the hierarchical sequence takes only six ($2\sqrt{N}$). Hierarchical organization has the additional advantage that more operators are learned that can be used in future instructions.

The single recall strategy overcomes the problems of multiple step projection. The recall and projection of a single instruction at a time does not require a time consuming introspection that suspends the agent's ongoing activity. Figure 3.14 illustrates this empirically, comparing the time

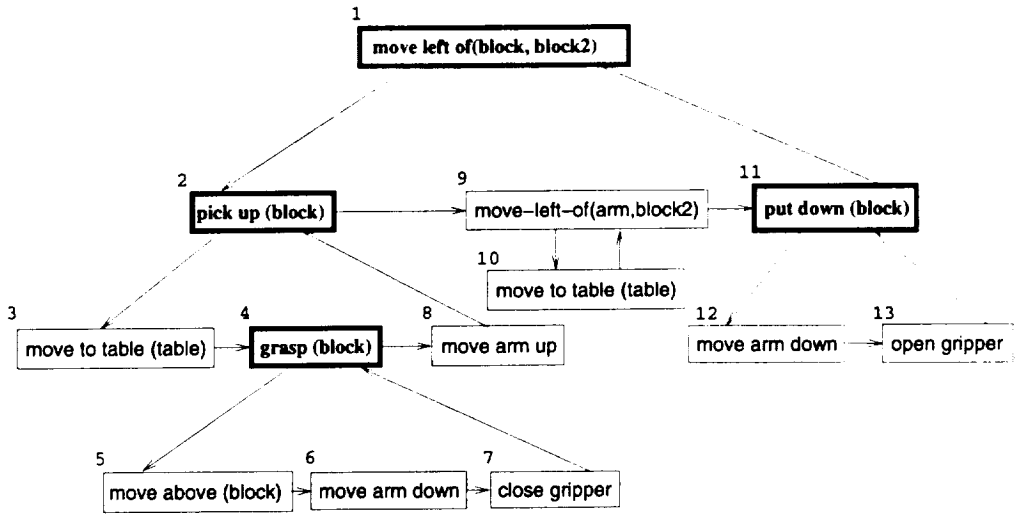



Figure 3.12: A graphical view of a hierarchical instruction sequence for `move-left-of(blk1,blk2)`. New operators are shown in bold.

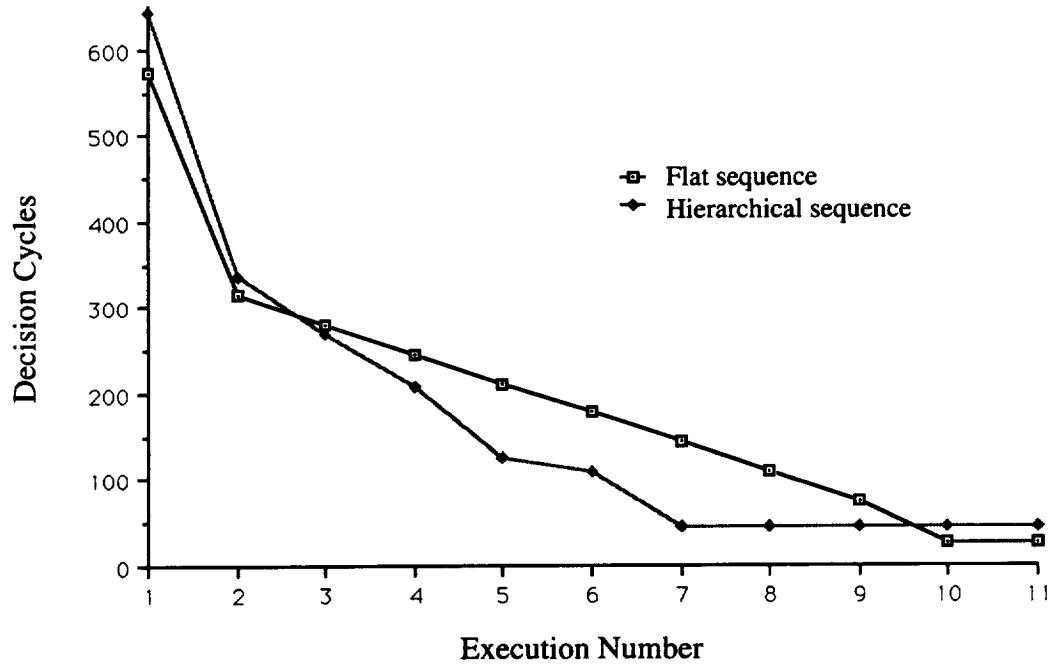



Figure 3.13: Decision cycles vs. learning trial for executing `move-left-of(blk1,blk2)`, comparing a flat instruction sequence to a hierarchical sequence (using the single recall strategy).

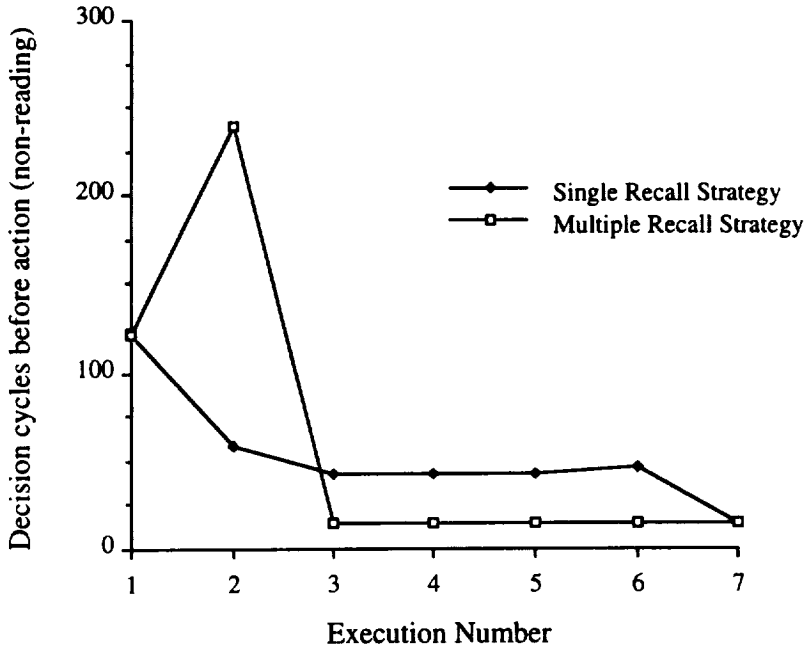



Figure 3.14: Decision cycles before external action vs. learning trial while learning "pick up", comparing the single- and multiple-recall strategies.

(in decision cycles) before an external action is taken on each execution of "pick up" when using the single and multiple recall strategies. The entire recall and explanation time is proportional to the length of the instruction sequence in both cases, but in the single recall strategy, that time is interleaved with the execution of the instructions rather than fully taken before one execution. The single recall strategy also overcomes the problem of compounding of domain theory errors, by beginning the projection of each instruction from the current state of the world after external execution of the previous instructions. Thus the beginning state of each projection correctly reflects the effects of the previous operators in the implementation sequence. The single and multiple recall strategies are actually endpoints of a continuum of strategies involving recall of any intermediate number of instructions.

## 3.5 Extending a procedure to a new situation

In addition to unknown procedures, an instructable agent may be asked to perform a known procedure in an unfamiliar situation – one from which the agent does not know how to successfully perform the procedure. An example is contained in the instructions for "Pick up the red block" shown in Figure 3.3, when the agent is asked to "Move above the red block." This is one of the agent's primitive operators: it knows the operator's goal concept (having its hand above the object), and it knows how to perform the operator *when its arm is raised.* The arm being raised is a precondition for the operator, from the perspective of the agent's current knowledge. However, in this case the arm is lowered, and so the agent does not know how to perform the instruction.

One option the agent could follow would be to search; i.e., to apply a weak method such as means-ends analysis. In this example, the reasoning would be fairly easy: the agent would simply need to figure out how to reach a state in which the arm is raised. In other cases, when the current state differs significantly from states where the instruction's "preconditions" are met, the search involved could be costly. In any event, since the goal of Instructo-Soar is to investigate the use of instruction, the Instructo-Soar agent always asks for instruction when it reaches an impasse in task

**If** the goal is `move-arm(destination (above <obj>))`, and
`<obj>` is sitting on a table `<table>`, and
the robot is docked at `<table>`, and
the arm is lowered
**then** propose operator `move-arm(up)`.

Figure 3.15: The operator proposal rule learned from the instruction "Move up" that extends the agent's ability to move above things to a new situation.

performance, rather than searching.[1]

Thus, the agent is told to "Move the arm up," and (as usual) attempts to explain this instruction through forward internal projection. The agent starts the projection from the current state, in which the robot is docked at the table with the red block on it, and the arm is down. The first step is to move the arm up within the projection. At this point, the implementation of "move above" applies, moving the arm above the red block. From this explanation of how moving up enables moving above, the agent learns the operator proposal rule shown in Figure 3.15. This rule extends the agent's knowledge of the "move above" operator to apply in the new situation. (The agent would have learned a similar rule if it had solved the problem through search rather than receiving instruction.)

"Move above" may be extended further through more instruction. For instance, the agent could be asked to "move above" before it is docked at the table the object is on, and learn to first move to that table. Extending an operator to a new situation could also require multiple steps. If multiple steps are used, they can be recalled and explained using any of the three recall strategies described in the previous section.

Newly learned operators may require extension to new situations as well. This is because when the agent learns the general implementation for a new operator, it does not reason about all *possible* situations that the operator might be performed in, but limits its explanations to the series of situations that arises during the actual execution of the new operator while it is being learned. In the example given in Appendix A, after learning to pick up things, the agent is asked to pick up a block while holding another block. The instructor must tell the agent to first put the held block down, which then allows the agent to pick up the other block. The agent learns to put down what it is holding before trying to pick up something else.

Notice again the flexibility afforded by allowing the instructor to command an operator at any time, regardless of the agent's current knowledge (requirement 7 of Table 3.1). The instructor does not have to keep track of what procedural knowledge the agent has, or avoid certain commands in certain situations; rather, instructions can be used to organize and extend tasks as the instructor wishes.

## 3.6 Dealing with incomplete underlying domain knowledge

All of the general implementation learning described thus far in this chapter depends on the agent being able to form complete explanations of each instruction using its prior domain knowledge. What if the domain knowledge is incomplete, making explanation impossible? For instance, what if the agent does not know about some of the key effects of its primitive operators?

As described in Chapter 2, a missing piece of knowledge, $M_K$, anywhere within the implementation sequence for a procedure being learned or extended, can result in an incomplete explanation. For implementation sequences consisting of multiple operators, pinpointing what knowledge is missing is an extremely difficult credit assignment problem (sequences known to contain only one operator, however, are a more constrained case, as described in the next chapter). In general, an

[1] Nothing in Instructo-Soar *precludes* the use of search or knowledge from other sources, however. In fact, one of Soar's strengths is integrating knowledge from different sources.

explanation failure that is detected at the end of the projection of an instruction sequence could be caused by missing knowledge about any operator in the sequence.

Thus, it is difficult to make a heuristic guess at the missing knowledge $M_K$, and it is unknown whether that knowledge will be acquired at some future point. Since the explanation cannot be completed without $M_K$, the agent's course of action is to abandon explanation and instead learn the knowledge $I_K$ (here, operator proposal knowledge, as in Figures 3.7 and 3.15) from each instruction $I$ directly, without the support of an explanation (option 2.1 from Table 2.4). That is, each proposal rule $I_K$ is learned inductively, by applying a set of heuristics and then allowing the instructor to alter and eventually verify the resulting knowledge.

This performance is best illustrated by an experiment in which all of Instructo-Soar's knowledge of secondary operator effects (frame axiom type knowledge) for the primitive operators are removed before instruction. For example, although the agent knows that closing the hand causes the gripper to have **status closed**, it no longer knows that closing the hand when directly above a block causes the block to be held. Secondary effects knowledge is used when internally projecting the effects of operators. A real agent is more likely to lack this type of knowledge (e.g., knowledge of some rare secondary effect of an action) than to lack basic knowledge about its primitive operators, such as their primary effects or termination conditions.

With this knowledge removed, the agent is taught a new procedure, such as to pick up the red block. After the first execution, the agent attempts to recall and explain the implementation sequence as usual. However, since secondary operator effect knowledge is missing (e.g., the knowledge that closing the gripper causes the block to be held), the forward internal projection used to explain the instruction sequence does not achieve the goal of the new operator (e.g., the block is not picked up at the end of the projection, since the agent's knowledge did not indicate it is held). The inability of the projection to achieve the new operator's termination conditions constitutes an explanation failure, and the agent records (by learning a chunk) the fact that this procedure's instructions cannot be explained.

Later, the agent is again asked to perform the procedure from a similar initial state. Since the agent has not yet learned a general implementation for the procedure, the past instructions are recalled. However, the agent also recalls that explaining the instructions failed in the past. Thus, rather than trying to explain each instruction $I$, the agent uses heuristics to induce proposal knowledge $I_K$ directly, and then allows the instructor to alter and verify this proposal.

For instance, in the "pick up" example, the agent first recalls the command to move to the yellow table. The general knowledge $I_K$ to be learned from this command is an operator proposal rule like that shown in Figure 3.7. Operator proposal rules propose an operator $OP$ (e.g., **move-to-table(<yellow-table>)**) in a state $S$ to achieve a goal $G$ (e.g., **new-op-14(<red-block>)**). To induce such a rule the agent must induce the conditions on $G$ and $S$ under which to propose $OP$. For the goal $G$, Instructo-Soar uses the simple heuristic that its name and the types of objects filling its slots are the key features. Here, the name is **new-op-14** (the name generated for the "pick up" operator); and there is a single slot, filled with an object that **isa block**. To determine the conditions of the state, Instructo-Soar employs two simple heuristics:

- **OP-to-G-path.** For an object $O1$ attached to $OP$ (filling a slot, e.g., **object**), and an object $O2$ attached to $G$, compute the shortest path of relationships between $O1$ and $O2$ of length (heuristically) less than 3. Each of the features that appears along the path are considered important conditions.

  This heuristic captures the intuition that if $OP$ involves some object, its relationship to the objects that are relevant to the goal $G$ may be important. The heuristic's operation for "move to the yellow table" is shown in Figure 3.16. As the figure indicates, there is a path between the red block, the object of $G$ (pick up the red block), and the yellow table, the **destination** of $OP$ (move to the yellow table). The path contains the feature that the block is **on** the table, which is added to the inferred set of conditions for proposing $OP$.

- **OP-features-unachieved.** Each termination condition of $OP$ that is *not achieved* in $S$ (the state before $OP$ is performed) is considered an important condition.

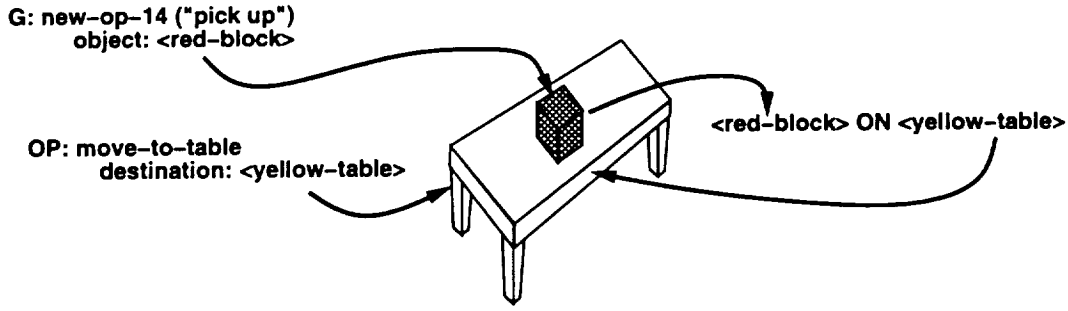



Figure 3.16: The use of the **OP-to-G-path** heuristic, with $OP$ "move to the yellow table," and $G$ "pick up the red block."

---

*So I'm guessing the conditions for doing "move to the table"*
*when your goal is "pick up the block" are:*
*the robot is not docked at the table*
*the block is sitting on the table*
*Is that right?*
> `Right.`

---

Table 3.5: Instructo-Soar's interaction with its instructor in inducing an operator proposal rule.

This heuristic captures the intuition that all of the primary effects of $OP$ (as encoded by its termination conditions) are probably important; therefore, it matters that they are not achieved when $OP$ is chosen to be performed. For the move to the yellow table instruction, $OP$'s primary effect is that the robot ends up docked at the table. Thus, the fact that the robot is *not* initially docked at the table is added to the inferred set of conditions for proposing $OP$.

Once the heuristics induce a set of conditions for proposing $OP$, the results are presented to the instructor, who can alter and eventually verify them. The interaction in this case is shown in Table 3.5. In this example, the heuristics have actually identified a reasonable set of features, so the instructor simply verifies the result. In other cases, the set of features must be altered. The operator proposal rule learned from this exchange is shown in Figure 3.17. It is similar to the rule learned from explanation (Figure 3.7), but only applies to picking up a **block**, and does not stipulate that object must be **small**.

Identifying the relevant conditions for selecting an operator can be difficult for the instructor, who must consider interactions between the conditions and the operator's effects in all possible situations. Of course, this is the same difficulty faced by a programmer when trying to program the agent for new tasks. Instructo-Soar allows this problem to be avoided when its domain knowledge is sufficient, and aids the instructor through the use of heuristics to help identify relevant conditions when its domain knowledge is insufficient.

After its proposal rule is learned, the "move to table" operator is applied in the external world. However, the agent still has not achieved the "pick up" goal, and so an impasse is reached, and the agent recalls the second instruction in the sequence, "Move above the red block." A proposal rule for this instruction is induced in the same way; the interaction is shown in Table 3.6. In this case, the instructor adds a condition: the robot must already be docked at the appropriate table before doing the "move above" action. If the instructor overlooks this condition, then both the "move to the table" and "move above" actions will be proposed simultaneously during future executions of the "pick up" procedure, requiring more instruction to learn which is preferred.

**If** the goal (higher-level operator) is **new-op-14** with **object** **<obj>**, and
**<obj>** is a **block**, and
**<obj>** is sitting on a table **<table>**, and
the robot is not docked at **<table>**
**then** propose operator **move-to-table(<table>)**.

Figure 3.17: The operator proposal rule learned for "Move to the yellow table" by direct induction, when domain knowledge is insufficient to explain the instruction.

---

*So I'm guessing the conditions for doing "move above the block"*
*when your goal is "pick up the block" are:*
*the arm is not above the block*
*Is that right?*
**> The robot must be docked-at the yellow table.**
*So I'm guessing the conditions for doing "move above the block"*
*when your goal is "pick up the block" are:*
*the arm is not above the block*
*the robot is docked at the table*
*Is that right?*
**> Right.**

---

Table 3.6: Induced conditions for an operator proposal can require alteration.

The process of learning a proposal rule and then executing the proposed operator occurs for each of the instructions, allowing the agent to learn a full, general implementation for the "pick up" operator. However, this implementation is not as correct as that learned from explanation; e.g., it applies to any block instead of to any small object. In a more complex domain, inferring general implementations would be even less successful. Psychological research shows that subjects' learning when given procedural instructions degrades if they do not have a domain model [Kieras and Bovair, 1984].

The process of directly learning operator proposal rules allows the agent to overcome incomplete domain knowledge, but is burdensome to the instructor, who must examine and alter/verify the conditions of proposing each operator. An incomplete explanation for a procedure probably indicates that some effect(s) of one of the operators in the procedure's instruction sequence is unknown. Thus, one alternative (which has not been implemented in Instructo-Soar) would be for the agent to deliberately observe the effects of each operator in the sequence as it performs them, comparing its observations to the effects predicted by its domain knowledge. Any differences that are noticed would allow the agent to induce new knowledge about operator effects that could allow the explanation of the procedure to be completed. Learning about the effects of operators from observation has been explored by a number of researchers (e.g., [Thrun and Mitchell, 1993; Pazzani, 1991b; Shen and Simon, 1989; Carbonell and Gil, 1987; Sutton and Pinette, 1985]). Instructo-Soar's learning of termination conditions for a new operator amounts to learning by observing the effects of a sequence of actions; learning operator effects from observation could be added as a variation of the same process.

Another alternative would be to learn about the unknown operator effects through instruction. Instructo-Soar is able to learn about the effects of operators through instruction in some cases, as described in the next chapter. However, because an explanation of a multiple-step procedure may fail because of missing knowledge about any effect of any of its steps, it is impossible in general for

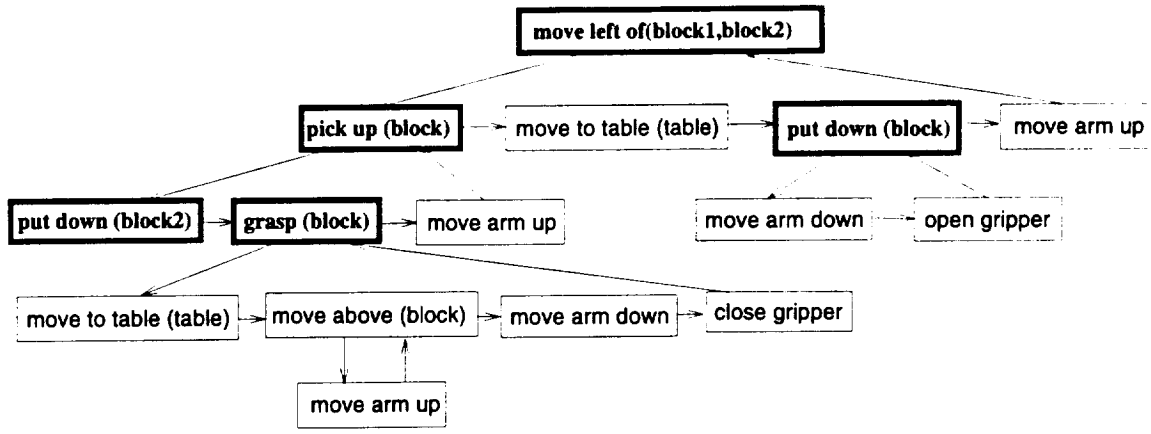



Figure 3.18: A hierarchy of operators learned by Instructo-Soar. Primitive operators are shown in light print; learned operators are in boldface.

the instructor (or the agent) to pinpoint what knowledge is missing based on this kind of explanation failure. Since identifying the unknown knowledge would be so difficult for the instructor, when the explanation of a multiple-step procedure fails, the agent resorts to directly inferring proposal rules rather than asking for further instruction.

## 3.7 Conclusion

This chapter has described Instructo-Soar's learning of completely new procedures, and the extension of known procedures to apply in new situations, from tutorial instructions. The agent supports flexible, hierarchical instruction, because at any point the instructor can request either a known procedure, an unknown procedure, or a known procedure that the agent does not know how to perform in the current state. This can lead to multiple levels of embedded instruction.

Newly learned operators may be specified in instructions for later operators, leading to learning of operator hierarchies. For example, a hierarchy of operators learned by Instructo-Soar is shown in Figure 3.18 (a subset of the learning described in Appendix A). Learning of procedural hierarchies has been identified as a fundamental component of children's skill acquisition from tutorial instruction [Wood *et al.*, 1976]. In learning the hierarchy of Figure 3.18, the agent 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). Because Instructo-Soar supports flexible interaction for commands, the hierarchy can be taught in many different ways. 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.

Learning of new procedures involves both inductive and analytic learning. In addition to learning from successful explanations (analytic learning), this chapter contains examples of all four of the options for dealing with incomplete explanations described in Chapter 2 (Table 2.3). From the initial execution of a new operator, the agent learns a rote, overspecific implementation sequence (option 1.1). At the end of the first execution, termination conditions for the new operator are induced (option 2.1), based on the difference between initial and final states, and feedback from the instructor (option 2.2). Later, the implementation sequence is moved to the proper level of generality through situated explanation, in which the agent recalls the instructions and explains to itself using forward internal projection how each instruction leads to completion of the new operator (successful explanation). If the agent's underlying domain knowledge is insufficient to successfully explain the instructions for a procedure, Instructo-Soar resorts to a strategy in which it directly induces operator proposal knowledge from each instruction (option 1.2). This can lead to high quality learning in simple cases, but in general the learning will be of lower quality than learning

| | Requirement | Met in Chapter 3 |
|---|---|---|
| 1. | General learning from specific cases | √ |
| 2. | Fast learning (each task instructed only once) | √ |
| 3. | Maximal use of prior knowledge | √ |
| 4. | Incremental learning | √ |
| 5. | Ability to learn all types of knowledge used in task performance | |
| | a. state inference | |
| | b. operator proposal | √ |
| | c. operator preferential | |
| | d. operator effects | |
| | e. operator termination | √ |
| 6. | Agent-initiated instruction | √ |
| 7. | Flexibility of knowledge content | |
| | a. full flexibility for action commands | √ |
| | b. others as appropriate | |
| 8. | Situation flexibility | |
| | a. implicitly situated | √ |
| | b. explicitly situated: hypothetical state | |
| | hypothetical goal | |

Table 3.7: The transfer and interaction requirements of tutorial instruction to be met, and those met thus far in the description of Instructo-Soar.

from explanations; in addition, the burden on the instructor is increased.

Table 3.7 indicates the transfer and interaction requirements of tutorial instruction that have been met at this point in the description of Instructo-Soar. Chapter 4 completes the description of Instructo-Soar by describing its use of explicitly situated instructions, and its coverage of each of the outstanding knowledge types remaining to be covered in Table 3.7.

# Chapter 4

# Instructo-Soar: Towards a Complete Instructable Agent

This chapter describes Instructo-Soar's coverage of each of the remaining requirements for a theory of tutorial instruction shown in Table 3.7. In particular, it discusses Instructo-Soar's ability to learn state inference knowledge, operator preferential knowledge, and knowledge about operators' effects, and its handling of a variety of types of explicitly situated instructions. Demonstrating these capabilities also reveals the interaction supported by Instructo-Soar for instructions other than action commands.

The agent's capabilities are demonstrated primarily by a series of selected examples. Although each example demonstrates a different capability, the chapter is organized such that the instructions given to the agent in the course of the chapter form an overall instruction scenario. The instructions build on one another in the sense that knowledge that the agent learns from earlier examples is sometimes used in later examples.

The examples primarily involve teaching the agent about the electromagnet and the light in its domain. At the outset, the agent knows very little these objects; it can perceive their visible properties, but does not know how to control them or (in the magnet's case) about their effect on other objects. It is assumed that the agent has already been taught procedures for pushing a button, for dimming and brightening the light, and for grasping objects, using the techniques described in the previous chapter. By the end of the instruction scenario of this chapter, the agent has learned how to control the magnet, how to turn on the light and keep it in a preferred status (on and dim), and enough knowledge of the effects of magnetism that it can pick up metal objects by using the magnet.

## 4.1 Learning from hypothetical states

Recall that the *situation* an instruction applies to includes a state, a goal (a higher level operator to be achieved), and an expectation about the instruction's effect at achieving the goal (either success, immediate success, failure, or immediate failure). The instructions in this section are explicitly situated; each explicitly specifies a hypothetical state that it is meant to apply to, rather than applying to the current world state. Instructo-Soar constructs a hypothetical state based on the semantic content of an explicitly situated instruction in a straightforward way; it includes in the state each of the objects, properties, and relationships mentioned in the antecedent of the instruction, as well as any objects that are needed to carry out the consequent. Once the hypothetical state is constructed, the agent attempts to explain and learn from the instruction, by using the constructed hypothetical state as the beginning state for an internal projection.

### 4.1.1 Learning state inference knowledge

State inference knowledge is knowledge about object properties and relationships that can be inferred based on the properties and relationships that currently hold. In Soar, this knowledge takes

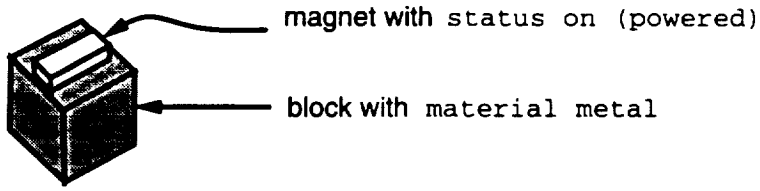



Figure 4.1: The partial hypothetical state built from the conditional state inference instruction, "If the magnet is powered, and directly above a block, and the block is metal, then the magnet is stuck to the block."

**If** the state has objects **<magnet>** and **<block>**, and
**<magnet> isa magnet** with **status on**, and
**<block> isa block** with **material metal**, and
**<magnet>** is **directly-above <block>**,
**then <magnet>** is **stuck-to <block>**.

Figure 4.2: The state inference rule learned from a conditional instruction.

the form of productions that test state conditions and produce new state properties and relations, without reference to any operator. These productions produce their results immediately whenever the state meets their conditions, and retract their results whenever their conditions no longer hold.

Instructo-Soar learns state inference rules from three different kinds of instructions:

1. Simple, specific statements; e.g., "The grey block is metal."
2. Simple, generic statements; e.g., "White magnets are powered"; "Red magnets are off."
3. Conditional statements; e.g., "If the magnet is powered, and directly above a block, and the block is metal, then the magnet is stuck to the block."

State inference instructions may introduce new properties or relations that were previously unknown to the agent. Thus, these statements can be used to extend the representation vocabulary that the agent uses to represent its world. For example, the relation **stuck-to** is not known to the agent before it is given the conditional instruction (3) above. Similarly, the agent can learn to associate new features with particular objects. For example, through the generic statements (2) above, the agent learns that magnets have a **status** – either powered or not.

The agent recognizes that an instruction communicates a state inference by the lack of any action being mentioned in the instruction. To learn state inferences, the system begins by constructing a hypothetical state containing the objects and properties mentioned in the statement. For simple statements like the examples above, the object mentioned before the copular verb is included in the hypothetical state; e.g., the grey block in example (1). Indefinite references in the instruction produce generic objects in the hypothetical state; definite references produce specific objects (with unique identifiers) from the current world state. For conditionals like (3), each of the conditions in the antecedent specifies a part of the hypothetical state. The hypothetical state constructed from (3) is shown in Figure 4.1.

After creating the hypothetical state, the agent simply asserts that the concluded property or relationship specified in the instruction holds in that state. For simple statements, the agent asserts the property specified after the copular verb (e.g., **material metal**); for conditionals, it asserts the consequent (e.g., the magnet is **stuck-to** the block). The assertion causes Soar to create a chunk that creates the property based on the conditions in the hypothetical state – a state inference rule. The chunk learned from the conditional instruction (3) above is shown in Figure 4.2.

Currently, Instructo-Soar does not use an explanation process to learn state inferences, since the focus of this work has been on learning procedural knowledge. An extension would be to try

**If** the goal is **happiness**, and
**<light> isa light** with **id light1** and **status off**
**then** propose operator **switch-status(object <light>, status on)**.

Figure 4.3: The operator proposal rule learned from the conditional instruction "If the light is off then turn on the light."

to explain why an instructed inference holds in the hypothetical state, possible resulting in more general learning.

### 4.1.2 Learning operator proposal knowledge from hypothetical states

Consider the following instructions:

```
> If the light is off, then turn on the light.
Why?
   > Trust me.
```

This conditional instruction indicates an operator that is to be proposed under a particular set of state conditions. It applies to a hypothetical situation with a hypothetical state specified by the instruction's antecedent. The goal of the hypothetical situation is the agent's top level goal, **happiness**, since the agent is not pursuing any other goals.

The agent creates a hypothetical state from the conditional instruction that contains a light that is off. Starting from this state, the agent attempts to explain the instruction by internally projecting the action of turning on the light.[1] However, turning on the light within the projection does not result in any increase of happiness; the agent does not understand why having the light on should make it any happier. Thus, the explanation of the instruction cannot be completed.

Failure to complete an explanation indicates missing knowledge. Here, because the instruction involves a single action to be taken under specific state conditions, the agent makes the heuristic guess that the missing knowledge does not involve further action (operators), but instead involves knowledge about the state. Thus, the agent asks for further instruction to gain that knowledge (option 2.2 from the set of strategies for dealing with incomplete explanation given in Table 2.4).

In this case, the instructor declines to give more instruction, by saying **Trust me.** This instruction indicates that the agent will not be given any further information, and must induce the reason that turning on the light should be successful (that it should cause **happiness**) based on its current knowledge. Since the agent does not know what features of the hypothetical state with the light on cause happiness, it takes the conservative approach of assuming that *all* the features of the hypothetical state must be relevant. Here, the state includes only the light that is on, so the agent asserts that when the light is on, **happiness** is met.

Asserting that the state with the light on achieves **happiness** completes the explanation of the "Turn on the light" action, and allows the agent to learn the desired operator proposal rule for the action, shown in Figure 4.3. The proposal rule applies to a specific light, **light1**, because of the definite reference to the light in the original instruction.

Consider a second example:

```
> If the light is bright then dim the light.
Why?
   > Dimmed lights use little electricity.
   > Bright lights use much electricity.
```

[1] The agent actually has not yet learned *how* to turn on the light (this will be learned later, as described below), but it does know the termination conditions of this operator, and so can simulate the primary effect, switching the light's status from **off** to **on**.

**If** the state has object **<light>**, and
**<light > isa light** with **brightness dim**,
**then <light>** has **electricity-consumption low**.

Figure 4.4: The state inference rule learned from "Dim lights use little electricity."

**If** the goal is **happiness**, and
**<light> isa light** with **electricity-consumption large**
**then** propose operator **new-op-47(object <light>)**.

Figure 4.5: The operator proposal rule learned from the conditional instruction "If the light is bright then dim the light." **New-op-47** is the procedure learned previously to dim the light.

Again, the agent builds a hypothetical state (with a bright light) and then projects the instructed action (dimming the light). In this example, the instructor chooses to give instructions to explain why dimming a bright light leads to happiness. From the instruction "Dimmed lights use little electricity," the agent learns a state inference rule (as described in the previous section), shown in Figure 4.4. This rule applies to the final state of the projection, with the light dimmed. When told "Bright lights use much electricity," the agent learns a similar rule that applies to the initial state of the projection, with the bright light. This triggers a part of the agent's prior knowledge about **happiness**. The agent knows that reducing the use of an expensive resource like electricity increases happiness. Thus, **happiness** is achieved in the internal projection, completing the explanation of the original instruction, and the agent learns the operator proposal rule shown in Figure 4.5. This rule generalizes the original instruction slightly, due to the explanatory instructions; it applies to any light that is consuming a large amount of electricity, rather than only the specific bright light referred to in the instruction.

### 4.1.3 Learning about contingencies

The conditional instructions in the examples above were given when the agent was not pursuing any particular goal. Explicitly situated instructions may also be given while the agent is learning to perform some procedure. Such instructions indicate a contingency to be followed when the procedure is performed in a situation different from the current situation. The ability to teach contingencies allows the instructor to teach the agent about alternative situations, including situations that may be rare or dangerous, thus overcoming one of the shortcomings of implicitly situated instructions. Contingency instructions are very common in tutorial discourse; Ford and Thompson [1986] found that 79% of the conditional statements in an instruction manual communicated contingency options to the student.

Consider this interaction between the agent and the instructor:

**> Grasp the blue block.**
*That's a new one for me. How do I do that?*
**> If the blue block is metal then grasp the magnet.**

The agent is being taught to grasp a block that is not metal. However, the instructor wants the agent to know that if the block *were* metal, a different course of action would be required.

The conditional instruction in this example ("If the blue block is metal then grasp the magnet") is similar to those in the previous section. From this instruction, the agent is expected to learn an operator proposal rule for grasping the magnet under appropriate conditions. The agent begins by determining the situation that the "grasp the magnet" instruction is meant to apply to. The

**If** the goal is **new-op-48(<block>)**, and
**<block> isa block** with **material metal**, and
**<magnet> isa magnet**
**then** propose operator **new-op-48(object <magnet>)**.

Figure 4.6: The operator proposal rule learned from the contingency instruction "If the block is metal, then grasp the magnet." (**new-op-48** is the new operator being learned for "grasp").

situation consists of the current goal ("Grasp the blue block") and a hypothetical state that is a variant of the current state, in which the blue block – the block being grasped – has the property **material metal**.

Starting from this hypothetical situation, the agent attempts to explain, by forward projection, how picking up the magnet allows the block to be grasped. If this projection were successful, the agent would learn a general operator proposal rule in the usual way. However, in this case the agent is missing much of the knowledge needed to complete the explanation. It does not know the goal concept of "Grasp" yet, or the rest of the instructions to reach that goal.

Since the instruction being explained is for a contingency, the rest of the instructions that the agent is given to "Grasp the blue block" may not (and in this case, do not) apply to the contingent situation, where the block is metal. In the normal grasp sequence, for instance, the agent learns to close its hand around the grasped object, but when grasping a metal object by using the magnet, the hand is closed around the magnet.

Since knowledge of how to complete grasping a metal object is needed to explain the contingency instruction, and the agent does not know when it might learn this missing knowledge, it chooses to abandon explanation. Instead, it directly induces the knowledge that can be learned from the contingency instruction (option 1.2 from Table 2.4). By applying the heuristics described in Section 3.6, the agent induces an operator proposal rule for "Grasp the magnet." In addition to the conditions generated by the heuristics, all of the conditions indicated in the antecedent of the instruction are included. Then, these conditions are presented to the instructor for alteration and verification:

*So I'm guessing the conditions for doing "grasp the magnet"*
*when your goal is "grasp the block" are:*
*the block is metal*
*Is that right?*
**> Right.**
*What next?*
**> Move to the yellow table.**
*Ok.* (does so) *What next?*
**> Move above the blue block.**
...

Once the instructor verifies the conditions, the agent learns the operator proposal rule shown in Figure 4.6. After this learning is completed, since the agent has not yet finished grasping the blue block, it continues to receive instruction for that task. Further contingencies can be specified to the agent at any point.

### 4.1.4 Learning to reject operators

This section describes learning to reject an operator – a type of operator preferential knowledge in the PSCM. The knowledge to reject an operator is communicated by a negative instruction that prohibits an action in a situation. Consider this example:

**If** a goal is **happiness**, and
**<block> isa block** with **color green**, and
**<gripper> isa gripper** that is **holding <block>**
**then** the state fails to achieve **happiness**.

Figure 4.7: The rule for recognizing a failure state, learned from an instruction prohibiting an action ("Never grasp green blocks"), and given no further explanation ("Trust me").

**If** an operator **new-op-48(object <block>)** is proposed, and
**<block> isa block** with **color green**, and
**then** reject operator **new-op-48(object <block>)**.

Figure 4.8: The operator rejection knowledge learned from an instruction prohibiting action ("Never grasp green blocks"), and given no further explanation ("Trust me") (**new-op-48** is the operator learned for grasping objects).

**> Never grasp green blocks.**
*Why?*
(a) **> Trust me.**
(b) **> Green blocks are explosive.**

A negative instruction is a command that is prohibited from applying to a hypothetical situation. When Instructo-Soar is given a negative instruction, it creates a hypothetical situation in which the prohibited action might be executed; in this case, a state with a green block that may be grasped. Since the hypothetical goal that this instruction applies to is not specified, and there is no current goal, the default top level goal of **happiness** is used. The goal expectation is **immediate failure**; that is, executing the prohibited action (grasping the green block) is expected to lead to an immediate lack of **happiness**.

From this hypothetical situation, the agent internally simulates the action of grasping the green block, to see if there is a negative (unhappy) result. However, an internal forward projection of grasp action results in what seems like a fine state. According to the agent's knowledge, it is still "happy" after the green block is grasped. Thus, the agent cannot understand why the action is prohibited. The agent deals with the incomplete explanation by asking for more instruction, guessing that the missing knowledge involves state inference.

As in the previous examples, the instructor may decline to give more instruction, by saying (a) **Trust me.** In response to this, the agent simply asserts that the final state of the projection, in which it is holding the green block, must be unhappy. This assertion causes the agent to learn a rule, shown in Figure 4.7, that recognizes the unhappy state. The failure to achieve happiness meets the goal expectation of **immediate failure**, completing the explanation of the original negative instruction. Based on this explanation, the agent learns an operator rejection rule for grasping green blocks, shown in Figure 4.8.

Alternatively, further instruction may be given to complete the agent's explanation of a negative instruction. For instance, the agent may be told (b) **Green blocks are explosive.** From this instruction, the agent learns a state inference rule: **blocks** with **color green** have **explosiveness high**. This rule adds **explosiveness high** to the green block that the agent had simulated grasping in the hypothetical situation. The agent knows that touching a highly explosive object may lead to an explosion; and that an explosion is a negative (unhappy) result. Thus, the agent concludes that the final state of the internal projection, in which it is grasping the green block, is an unhappy state, learning the rule shown in Figure 4.9. Reaching this negative result completes the explanation of the original instruction, and the agent learns an operator rejection rule that avoids grasping objects with **explosiveness high**, shown in Figure 4.10.

**If** a goal is **happiness**, and
**<obj>** has **explosiveness high**, and
**<gripper> isa gripper** that is **holding <obj>**
**then** the state fails to achieve **happiness**.

Figure 4.9: The rule for recognizing a failure state, learned from an instruction prohibiting an action ("Never grasp green blocks"), and given explanatory instruction ("Green blocks are explosive").

**If** an operator **new-op-48(object <obj>)** is proposed, and
**<object>** has **explosiveness high**, and
**then** reject operator **new-op-48(object <obj>)**.

Figure 4.10: The operator rejection knowledge learned from an instruction prohibiting action ("Never grasp green blocks"), and given explanatory instruction ("Green blocks are explosive").

Completing the explanation through an explanatory instruction, case (b), allows the agent to do more general learning than in case (a). The rules from case (b), shown in Figures 4.9 and 4.10, apply to any explosive object, not only green blocks. If the agent is later told, for instance, that **Blue blocks are explosive,** it will know to avoid grasping blue blocks:

> **Blue blocks are explosive.**
*Ok.* (the agent learns a state inference rule)
> **Grasp the blue block.**
*I cannot grasp the blue block. I know it will fail.*

In addition to learning to avoid the specific *action* of grasping certain objects, the agent has learned to recognize certain *states* that fail to reach its goals. These states should be avoided even if *grasping* is not explicitly mentioned. For example, consider the following exchange:

> **Move above the green block.**
*Ok.* (agent does it)
> **Move down.**
*Ok.* (agent does it)
> **Close the hand.**
*Ok.* (agent does it)
*This is a bad state to be in!* (agent opens its hand)

Since the agent does not think about the possible consequences of suggested actions *before* performing them, it does not realize before closing its hand around the green block that this action will lead to a bad state. However, after the action is taken, the rule that recognizes a bad state (from Figure 4.9) applies. Upon recognizing the bad state, the agent does two things. First, it selects an operator that will *reverse* the primary effect of the previous operator. The agent knows how to reverse primary effects for each of its primitive operators; here, it opens the hand, to get out of the bad state. Second, the agent takes the opportunity to learn about the operator that led to the bad state. It does this in its usual way; namely, it internally projects "close the hand," and sees that it leads to a failed state. From this explanation, the agent learns to reject the operator in the future when it will lead to the bad state, *before* performing it. The rule learned here is shown in Figure 4.11. Now the agent will avoid closing its hand around explosive objects:

> **Close the hand.** (when directly above the green block)
*I cannot close my hand. I know it will fail.*

**If** an operator **close-gripper** is proposed, and
**<object>** has **explosiveness high** and **size small**, and
**<gripper> isa gripper** that is **above-xyz <object>**)
**then** reject operator **close-gripper**.

Figure 4.11: Operator rejection knowledge learned when the agent closes its hand around an explosive object, leading to a bad state.

Notice the effect of the *situated* nature of Instructo-Soar's learning. The agent learns to avoid operators that lead to a bad state only when they *arise* in the agent's performance. Because of this, the agent momentarily arrives in the bad state. Its initial learning about the state is recognitional rather than predictive. Alternatively, when the agent first realized that (for instance) states where its gripper is holding an explosive object are to be avoided, it could have done extensive reasoning to determine every possible known operator that could lead to that state, from every possible previous state, to learn to reject those operators at the appropriate times. This reasoning would have been difficult, because of its unsituated nature; the agent would have to reason through a huge number of possible situations. In addition, whenever new operators were learned, the agent would have to reason about all the possible situations they could arise in, to learn if they could ever led to a bad state. Rather than this costly reasoning, Instructo-Soar simply learns what it can from its current situation. The theory makes the prediction that human learners also initially learn recognitional knowledge, and may get into bad states before learning to avoid the actions that led to them.

A more viable alternative for avoiding bad states than reasoning about all possible situations would be to think through (internally project) the effects of every action before taking it, to see if a bad state will result. This highly cautious execution strategy may be appropriate in dangerous situations. It is not appropriate in safer situations where the agent is under time pressure to perform. The agent could choose between more or less cautious execution strategies in different situations using knowledge (e.g., general domain knowledge, temporal knowledge). This is not currently implemented in Instructo-Soar.

From instructions that indicate a hypothetical state, then, the agent can learn state inference knowledge, operators proposal knowledge (based on state conditions, and as contingencies within a larger procedure) and operator rejection knowledge.

## 4.2 Learning from hypothetical goals

In addition to hypothetical states, instructions may explicitly specify the goal of the situation that they are intended to apply to. Within a single instruction, an explicitly specified goal takes the form of a purpose clause [DiEugenio, 1992], that indicates the purpose (goal) that an action is meant to achieve. Such instructions have the form "To do X, do Y" (e.g., "To turn on the light, push the button"). The basic knowledge to be learned from such an instruction is operator proposal knowledge for doing Y when the goal is to achieve X.

Instructo-Soar deals with purpose clauses by first mapping "do X" and "do Y" to operators, and then creating a hypothetical situation in which the operator derived from the purpose clause "do X" is the hypothetical goal. Since no state is specified, the state of the hypothetical situation is taken to be the current world state. The expectation about achieving the hypothetical goal is initially assumed to be **immediate success**; that is, the agent assumes that doing the specified action ("do Y") will lead to immediate achievement of the purpose clause goal ("X"). In DiEugenio and White's [1992] terms, the agent assumes that doing Y *generates* action X. This may be incorrect; doing Y may be just an initial step towards achieving X. However, in an analysis of instructional texts, DiEugenio [1992] found that fully 95% of instructions with purpose clauses expressed generation (action Y fully achieved action X).

**If** the goal is `switch-status(object <light>, status on)`, and
`<light> isa light` with `id light1` and `status off`, and
`<button> isa button` with `id button1` that has `color red`)
**then** propose operator `new-op-19(object <button>)`.

Figure 4.12: Operator proposal knowledge learned from an instruction specifying a hypothetical goal ("To turn on the light, push the button.") `New-op-19` is the operator learned previously for pushing a button.

### 4.2.1 Learning operator proposals from hypothetical goals

Once it is built, the hypothetical situation is used as the starting point for an internal forward projection of Y, to explain how it leads to achievement of X. If this projection succeeds, the agent learns an operator proposal rule proposing Y.

For instance, consider the instruction:

```
> To turn on the light, push the red button.
```

Assume for a moment that the agent knows pushing the button will toggle the light's status (in actuality the agent must learn this). When given the instruction above, the agent will form a hypothetical situation with a goal of achieving "turn on the light", a state like the current state, and an expectation that pushing the red button will cause immediate success at reaching the goal. When the agent internally projects pushing the button, the light will be toggled on, achieving the goal. Thus, the explanation of the instruction is successful, causing the operator proposal rule in Figure 4.12 to be learned.

### 4.2.2 Learning operator effects knowledge

However, in actuality the agent does not know the effect that pushing the button has on the light's status. After creating a hypothetical situation from "To turn on the light, push the red button," it internally projects pushing the button, but because of its lack of knowledge about the effects pushing the button, the light remains off within the projection. The explanation is incomplete. The incomplete explanation and the knowledge needed to complete it are shown in Figure 4.13.

When Instructo-Soar's explanation of a *sequence* of operators fails, the agent gives up on explanation and induces operator proposal rules for the sequence directly, as described in Section 3.6. The agent does not try to figure out what missing knowledge is needed to complete the explanation, because for a sequence of multiple operators, the credit assignment problem of determining what knowledge is missing is extremely difficult.

However, in this case, the agent is faced with the incomplete explanation of a sequence that contains only *one* operator. Because the sequence is only one operator long, and the agent knows what conditions that operator is supposed to accomplish (the termination conditions of the hypothetical goal to be achieved), the credit assignment problem is straightforward. The agent can make a good guess at the missing knowledge needed to complete its explanation (option 2.1 from the list of options for dealing with incomplete explanations, Table 2.4): it heuristically guesses that since the operator should achieve the hypothetical goal, its effect must be to produce the termination conditions of that goal. Thus, any termination conditions of the goal that are not met after the operator is projected are guessed to be effects of the operator that are unknown to the agent.

In this case, the termination condition for "turn on the light" is that the light have `status on`. After internally projecting "push the red button", the light's status is still `off`. Thus, the agent infers that an effect of "push the red button" must be to change the status of the light from `off` to `on`. This inference is presented to the instructor for verification:

*Does this mean that an effect of "push the button" is that:*

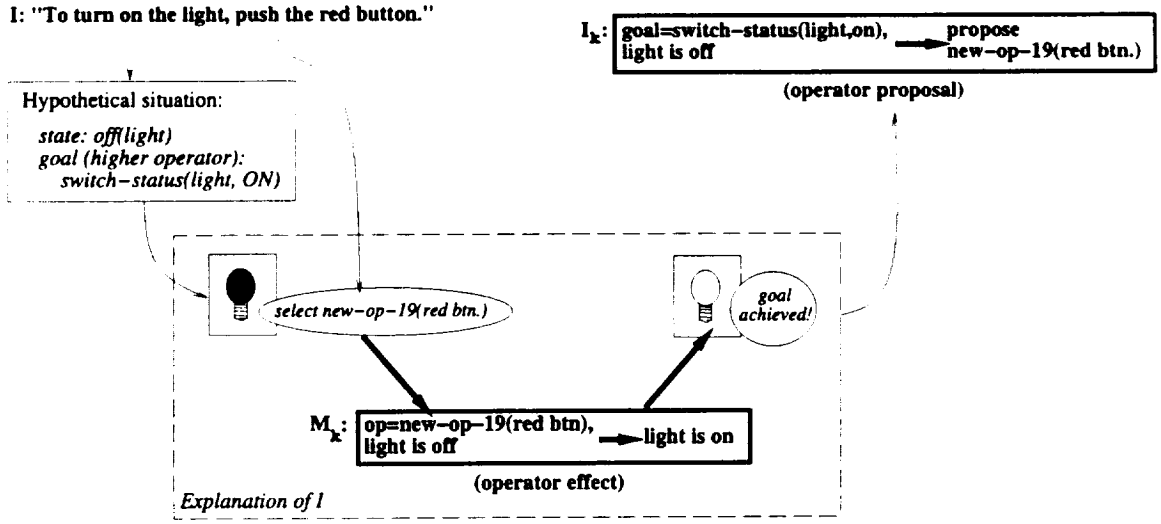



Figure 4.13: An incomplete explanation of an instruction $I$ with a hypothetical goal, and the knowledge $M_K$ needed to complete it. `New-op-19` is the operator learned for pushing the button.

*the light is now on*
*Is that right?*
`> Yes.`

The instructor verifies that pushing the button has the effect of turning on the light. However, this effect (and effects of operators in general) is conditional on particular features of the state that hold before the operator is executed. To guess at these conditions, Instructo-Soar uses two simple heuristics that operate as a very naive causality theory [Pazzani, 1991a] for determining the causes of the inferred operator effect. The **OP-to-G-path** heuristic (described in Section 3.6) includes any features on paths between objects attached to the operator and to the goal. In the current example, **OP-to-G-path** notices that the light and the red button are both on the same table. The **goal-conditions-unmet** heuristic includes each of the goal's termination conditions that were not met before the instructed operator was projected; this includes negations of each newly inferred effect of the operator in question. Here, **goal-conditions-unmet** adds that the light is not on before the button is pushed. The result is presented to the instructor for alteration and verification:

*I think that doing push the button causes:*
*the light to be on*
*under the following conditions:*
*the light is not currently on*
*the light is on the table*
*the button is on the table*
*Are those the right conditions?*
`> The button must-be red.`
*I think that doing push the button causes:*
*the light to be on*
*under the following conditions:*
*the light is not currently on*
*the light is on the table*
*the button is on the table*
*the button is red*
*Are those the right conditions?*

**If** the operator being projected is **new-op-19(object <button>)**, and
**<light> isa light** with **status Not on**, and is **on <table>**, and
**<button> isa button** that has **color red**, and is **on <table>**,
**then <light>** now has **status on**.

Figure 4.14: A learned operator effect of pushing the red button (**new-op-19** is the operator learned previously for pushing a button).

**> Right.**

Since the heuristics do not recognize that the specific button being pushed is important, the instructor added the condition that the button must be red (pushing the green button has different effects). Once the instructor verifies the conditions, the agent learns the operator effects rule shown in Figure 4.14. This rule will apply whenever the agent internally projects pushing the red button. In learning an operator effect, the agent has extended its basic domain knowledge.

After this effect is learned, the agent again mentally simulates pushing the red button, to explain the original instruction, **To turn on the light, push the red button.** The newly learned operator effect knowledge causes the light to come on and the goal to be reached, completing the explanation. This enables the agent to learn an operator proposal rule that proposes pushing the red button when trying to turn on the light, like that shown in Figure 4.12. The agent has acquired new knowledge at multiple levels; learning an unknown effect of an operator supported learning a proposal for that operator.

Currently, Instructo-Soar is only able to learn about operator effects through purpose clause instructions, as in this example. The instructor cannot, for instance, say "When you do X, it causes Y and Z to happen." This is a limitation of Instructo-Soar's mapping and interaction capabilities.

What if the inferred effect of pushing the red button was incorrect? For instance, what if rather than causing the light to come on, pushing the button simply activated a switch that could be thrown to turn on the light? If the instructor rejects the inferred effect as incorrect, Instructo-Soar realizes its knowledge it too incomplete to learn at multiple levels. Thus, the system resorts to directly learning the knowledge it can induce from the instruction (option 1.2 from Table 2.4) – namely, an operator proposal rule for pushing the button.

### 4.2.3 Learning about hidden/inferred operator effects

The operator effects rule learned in the previous example (Figure 4.14) applies only within the internal projection of the operator. The rule, which concludes that the light comes on when the red button is pushed, is not needed when the button is actually pushed externally, because the agent can just see that the light comes on. In this section, an example is presented in which the agent learns about an unknown effect of an operator, but the effect is not directly perceived, and the agent must *learn* to recognize it in actual execution.

Recall that the agent does not know about magnetism. The instructor would like to teach the agent to "grasp" metal blocks by using the magnet. To understand this kind of grasping, the agent must learn about the effect of moving a powered magnet into contact with a metal object.

To teach the agent about the effect of the magnet, the instructor first leads the agent to the state shown in Figure 4.15. Here, the robot is holding the magnet above the grey block, which is a large metal block, and the magnet is powered.

Now, the instructor tells the agent:

**> To grasp the grey block, move the arm down.**

As described in the previous sections, the agent forms a hypothetical situation with the goal of grasping the grey block. Starting from the current state, it forward projects moving the arm

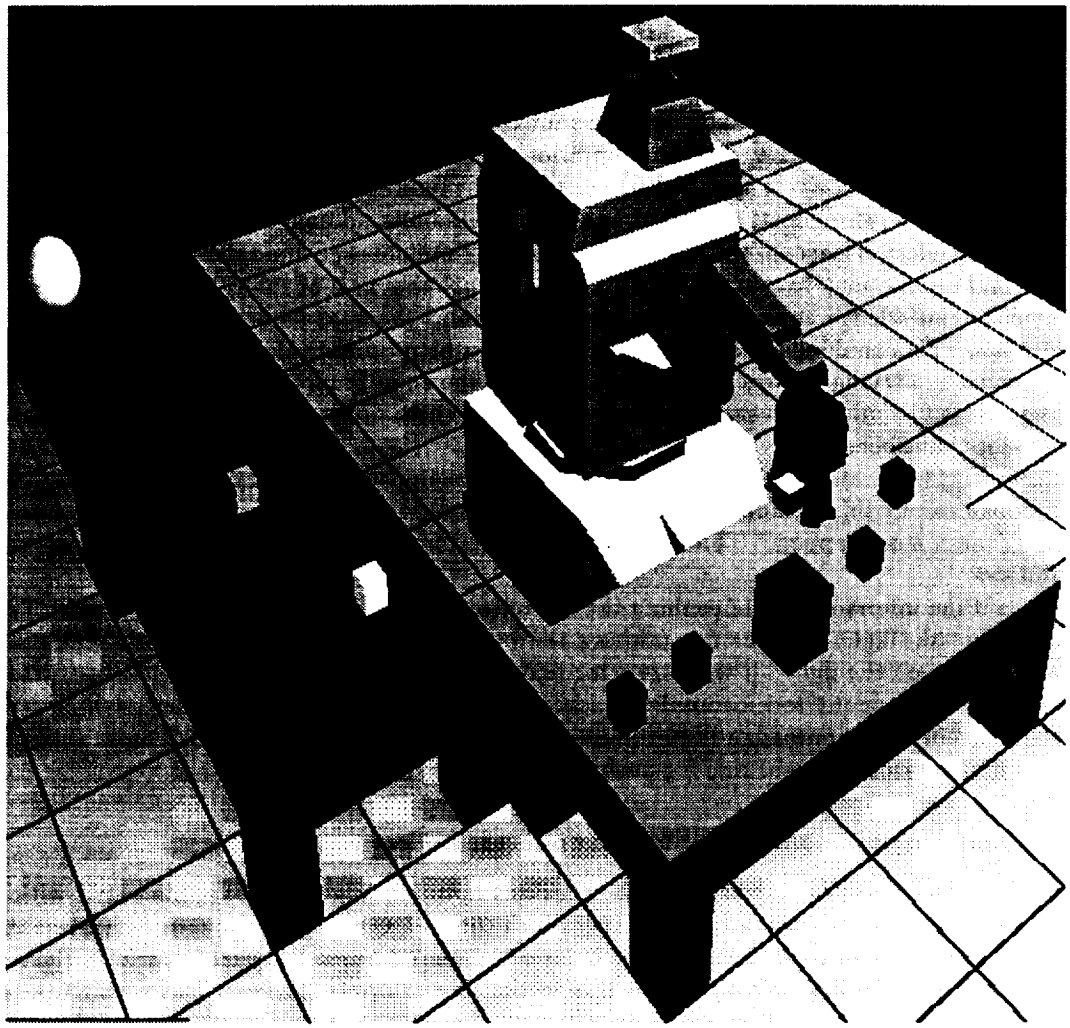


Figure 4.15: The initial state the agent is placed in for learning about grasping by using a magnet. The agent is holding the magnet, and the magnet is on.

**If** the operator being projected is **move-arm(down)**, and
**<gripper> isa gripper** that is **above-xy <block>**, and
**<gripper> NOT holding <block>**, and **<gripper> holding <magnet>**, and
**<magnet>** has **status on**, and
**<block>** has **material metal**,
**then <gripper>** is now **holding <block>**.

Figure 4.16: A learned operator effect of moving down when holding a powered magnet above a metal object.

down, expecting this to achieve the desired grasp. Of course, since the agent does not know about magnetism, this projection fails to cause the grey block to be grasped. Thus, the agent tries to infer some unknown effect of moving the arm down that would cause the grasp to be completed successfully (using the same method as in the previous example):

*Does this mean that an effect of lowering the arm is that:*
*the gripper is now holding the block*
*Is that correct?*
**Yes.**
*I think that doing lower the arm causes:*
*the gripper to be holding the block*
*under the following conditions:*
*the gripper is not holding the block*
*the gripper is above the block*
*the gripper is holding the magnet*
*Are those the right conditions?*
**> The magnet must-be powered.**
...
**> The block must-be metal.**
*I think that doing lower the arm causes:*
*the gripper to be holding the block*
*under the following conditions:*
*the gripper is not holding the block*
*the gripper is above the block*
*the gripper is holding the magnet*
*The magnet is on.*
*The block is metal.*
*Are those the right conditions?*
**> Right.**

From this exchange, the agent learns a new effect of moving the arm down. The rule that is learned is shown in Figure 4.16. Just as in the example of the previous section, learning this rule allows the agent to complete its explanation of how moving down succeeds at grasping the grey block; it achieves the termination conditions for "grasp", that the object being grasped is now held. From this explanation, the agent learns an operator proposal rule for moving the arm down to grasp a metal block, shown in Figure 4.17.

Next, the agent is asked to go ahead and grasp the metal block:

**> Grasp the grey block.**

The proposal rule in Figure 4.17 for moving down that was just learned applies, and the agent moves its arm down. After the arm is lowered, the state inference rule shown in Figure 4.2 applies to add the fact that the magnet is now **stuck-to** the block. However, the agent does not realize that it is now "holding" the block.

**If** the goal is **new-op-48(object <block>)**, and
**<block> isa block** with **material metal**, and
**<gripper> isa gripper** that is **NOT holding <block>**, and
**<gripper>** is **holding <magnet>**, and **above-xy <block>**
**<magnet> isa magnet** with **status on**, and
**then** propose operator **move-arm(down)**.

Figure 4.17: Operator proposal knowledge learned from an instruction specifying a hypothetical goal ("To grasp the grey block, move down.") **New-op-48** is the operator learned previously for grasping an object.

**If** the state has objects **<magnet>**, **<gripper>**, and **<block>**, and
**<magnet> isa magnet**, that is **stuck-to <block>**, and
**<gripper> isa gripper** that is **holding <magnet>**, and
**<block> isa block**,
**then <gripper>** is **holding <block>**.

Figure 4.18: A state inference rule that enables the agent to recognize a goal achieving state, for grasping a metal block.

Next, since success is expected at grasping after the arm is moved down, the termination condition for "grasp" (that the block be held) is tested. The condition does not hold in the current state, because the agent does not know it is holding the grey block. Although the agent learned to conclude "holding" when *projecting* the "move down" operator (the rule in Figure 4.16), this effect is not directly perceivable in the external world.

Thus, agent must learn to infer that it is **holding** the block based on other state conditions. It tries to induce a new state inference rule that will infer the needed goal conditions:

*So, can I infer that:*
*the gripper is holding the block*
*Is that right?*
**> Yes.**

If the instructor does not verify the inferred property, then the agent will assume that the goal (grasping the block) is not yet reached and simply ask for more instructions to reach it. In this case, the inference is correct. However, the agent does not know what state conditions cause the inferred property to hold. Thus, the instructor is asked:

*Why is the gripper holding the block now?*
**> Because the gripper is holding the magnet and**
**the magnet is stuck-to the grey block.**
*Ok.*

From this exchange, the agent learns a new state inference rule, shown in Figure 4.18. The rule allows the agent to recognize that the gripper is "holding" the grey block through use of the magnet.

From instructions that specify hypothetical goals, then, Instructo-Soar can learn operator proposal knowledge, and knowledge of both perceivable and inferred operator effects. The final category of learning to be discussed is learning of operator comparison knowledge.

## 4.3 Learning operator comparison knowledge

An operator comparison rule is a type of operator selection knowledge that expresses preference for one operator over another in a given situation, when both operators have been proposed. This

**If** the goal is **new-op-48(object <obj>)**, and
an operator **new-op-48(object <magnet>)** is proposed, and
an operator **move-to-table(destination <table>)** is proposed, and
**<obj>** has **size small**, and is **on <table>**, and
**<table> isa table**, and
**<robot> isa robot** that is **Not docked-at <table>**, and
**<gripper> isa gripper** that has **status open**,
**then** prefer operator **move-to-table** over **new-op-48**.

Figure 4.19: An operator comparison rule learned for grasping small metal blocks.

type of knowledge allows the agent to avoid impasses that arise because of ties between proposed operators.

Previously, the agent learned how to grasp small objects by using its hand. While teaching the agent to grasp, the instructor mentioned a contingency action: if the block being grasped is metal, the agent should begin by grasping the magnet. The rule learned for this contingency is shown in Figure 4.6.

What if the agent is asked to grasp a block that is both **small** and **metal**? In such a case *both* the contingency action – to grasp the magnet – and the regular action for grasping small blocks – to move to the table the block is sitting on – will be proposed. The agent has no further knowledge to select between these two options, and reaches an impasse because of the resulting tie.

To resolve this impasse, the agent must learn operator comparison knowledge that prefers one operator to the other (or, perhaps, a third operator that is better than either of the current choices). Thus, the agent asks the instructor for help:

**> Grasp the yellow block.**
*These possibilities come to mind:*
*move to the table*
*grasp the magnet*
*What should I do?*
**> Move to the yellow table.**
*Ok.*

The instructor indicates the preferred course of action, allowing the agent to learn an operator comparison rule preferring that option. To learn the rule, the agent recalls how the chosen option leads to the goal, and based on that explanation, asserts it as the preferred choice. The rule that results is shown in Figure 4.19.

Alternatively, the instructor could have specified some other action besides the two options listed. If this happens, the agent first explains the instruction in the usual way, to learn an operator proposal rule for it; and then learns operator comparison rules like that in Figure 4.19 to prefer the instructed operator to each of the other proposed possibilities.

Another alternative is that the possible options are equally good choices. The instructor can indicate this by saying "Choose either one" when asked which option should be followed. From this instruction, the agent will learn an operator comparison rule like that of Figure 4.19, but instead of preferring one operator over the other the rule will indicate that they are *indifferent*. This knowledge will cause Soar to choose one of the options at random.

Unfortunately, in the current example, after the agent moves to the yellow table, there are again two possible actions proposed. The next step in grasping a small object is to move the arm above it. However, the rule that proposes grasping the magnet (Figure 4.6) still applies, proposing again that the magnet be grasped.

*These possibilities come to mind:*
*move above the block*

*grasp the magnet*
*What should I do?*
**> Move above the yellow block.**
*Ok.*

From this exchange, the agent learns another operator comparison rule similar to the one in Figure 4.19. This pattern continues until the final step of grasping the block is completed. Later, when the agent is again asked to grasp a small metal block, the operator comparison rules learned will apply to select the preferred method of grasping the block directly without use of the magnet.

There are two weaknesses to the agent's learning of operator comparison knowledge. First, as this example illustrates, the instructor can be forced to indicate a preference for each step needed to complete a procedure, rather than simply choosing between overall methods. That is, the instructor cannot say "Use the method where you grab the block with your hand, instead of using the magnet." This is because knowledge about each step in a procedure is stored and accessed independently, as separate proposal rules, rather than grouped together as a method. This local knowledge organization improves flexibility and reactivity, allowing the agent to combine steps from different methods as needed based on the current situation. However, a higher level grouping of steps associated with particular methods – a more global organization – would allow more reasonable instruction for selecting between methods.

The second weakness is that although the agent uses its explanation technique to explain the selection the instructor makes, it does not explain why that selection is *better* than the other possible choices. When a tie arises, both choices "work" in that they are viable steps toward achieving the goal. Thus, preferences between them typically take the form of global preferences; e.g., "Prefer actions that lead to faster/cheaper goal achievement." Such a preference would give the agent a basis for explaining why one action is preferred over another. The preference is non-local because applying it requires aggregating information from multiple situations and actions – i.e., adding up the cost of alternative paths from the current situation to the goal. It is possible to encode such global control knowledge within the PSCM, but Instructo-Soar's instructional learning technique currently cannot handle instructions that specify non-local control preferences.

## 4.4 Conclusion

Instructo-Soar is an implementation of the theory of learning from tutorial instruction described in Chapter 2. The agent's architecture is the problem space computational model, as embodied by Soar. To learn from instruction, the agent uses situated explanation.

Instructo-Soar meets each of the requirements for an instructable agent that were specified as targets for the theory in Chapter 2. These requirements are summarized in Table 4.1. The agent uses a straightforward approach to the mapping problem, which has not been a major focus of this research. It provides a more complete solution to the interaction problem than previous systems, handling both implicitly and each type of explicitly situated instructions, and supporting fully flexible interaction for action commands. Finally, the agent meets each of the requirements of the transfer problem for extending incomplete knowledge through instructional learning. The agent is *complete* with respect to instructability of its knowledge types, being able to learn all of the types of knowledge that it uses in task performance from instruction.

| | Requirement | | Met? | Where described |
|---|---|---|---|---|
| 1. | General learning from specific cases | | √ | Chapters 3-4 |
| 2. | Fast learning (each task instructed only once) | | √ | Section 3.4 |
| 3. | Maximal use of prior knowledge | | √ | Chapters 3-4 |
| 4. | Incremental learning | | √ | Chapters 3-4 |
| 5. | Ability to learn all types of knowledge used in task performance | | √ | Chapters 3-4 |
| | a. state inference | | √ | Section 4.1.1 |
| | b. operator proposal | | √ | Sections 3.4, 4.1-2 |
| | c. operator preferential: | rejection | √ | Section 4.1.4 |
| | | comparison | √ | Section 4.3 |
| | d. operator effects | | √ | Sections 4.2.2-3 |
| | e. operator termination | | √ | Section 3.4 |
| 6. | Agent-initiated instruction | | √ | Chapters 3-4 |
| 7. | Flexibility of knowledge content | | √ | Chapters 3-4 |
| | a. full flexibility for action commands | | √ | Chapter 3 |
| | b. others as appropriate | | √ | Chapter 4 |
| 8. | Situation flexibility | | √ | Chapters 3-4 |
| | a. implicitly situated | | √ | Chapter 3 |
| | b. explicitly situated: | hypothetical state | √ | Section 4.1 |
| | | hypothetical goal | √ | Section 4.2 |

Table 4.1: The target requirements for a tutorable agent that are met by Instructo-Soar.

# Chapter 5

# Cognitive Properties

This chapter examines Instructo-Soar and the theory of instructional learning that underlies it from a cognitive perspective. It focuses on the behavioral properties and predictions that result when Instructo-Soar is viewed as a potential cognitive model for the task of learning procedures from tutorial instruction. Instructo-Soar's development has been strongly influenced by the PSCM and by Soar. Soar has been put forward as a candidate unified theory of cognition (UTC) [Newell, 1990]; thus, one key issue to examine is what kind of impact this unified theory has had on the model's development. Does the use of a unified theory provide positive constraint to the cognitive modeler? The chapter analyzes Soar's effect on the Instructo-Soar model, and in so doing, develops a general methodology for analyzing and comparing individual microtheories that have been created within unified theories of cognition.[1]

Specifically, the chapter addresses the following issues:

- What are the behavioral properties of Instructo-Soar as a model of learning procedures from instruction?
- How did Instructo-Soar's development within a unified theory of cognition (Soar) provide constraint to give rise to these behavioral properties? That is, what impact does being within Soar have on the model?
- What predictions about human behavior result from the properties of the model?
- How do these predictions compare to what is known of human behavior?
- How does the model compare to other cognitive models of similar tasks? To models developed in other unified theories of cognition?

Viewed as a cognitive model, Instructo-Soar makes a number of testable predictions about human learning, although the agent's performance does not cover the full range of interaction that occurs in real human tutorial learning. Unfortunately, there is not a great amount of data on the detailed course of human learning during tutorial instruction; most instructional research has focussed on macro-issues like the overall effectiveness of different instructional strategies. However, Instructo-Soar's behavioral predictions are found to be broadly consistent with that is known of human behavior. The analysis of Soar's impact on Instructo-Soar indicates that the properties of Soar have a large influence on properties and predictions of the model, showing how the use of a UTC can provide constraint in model building.

[1] This methodology for UTC-based microtheory analysis was developed with Craig Miller and John Laird [Huffman *et al.*, 1993a].

## 5.1 Instructo-Soar as a model of procedural learning from tutorial instruction

To focus the discussion, only the central functionality of Instructo-Soar will be examined as a cognitive model: the learning of new procedures from situated interactive instructions, described in Chapter 3. To recap the performance: Instructo-Soar exhibits two stages of learning behavior in learning a new procedure. Initial learning is rote and episodic in nature, and exhibits low transfer. Later executions result in high transfer, general learning, based on situated explanation of each instruction. That is, whenever the agent is given an instruction (or recalls one it was given in the past), it attempts to explain to itself why that instruction leads to achievement of its current goals. This explanation takes the form of an internal forward projection of the instructed action – the agent asks "What would happen if I did that?" If this projection results in successful goal achievement (or in failure if the instruction is to avoid an action), then Instructo-Soar learns a general rule capturing the key conditions under which the instructed action applies. A similar "learning by (simulated) doing" process applies when the agent is given an instruction that applies in a hypothetical situation; the agent first creates a hypothetical context, and then projects the instructed action, attempting to understand why it leads to achieving the agent's goals.

The main focus of research on Instructo-Soar has been on the transfer problem; that is, on the process of learning when given instruction. Thus, the primary behavioral properties of Instructo-Soar that will be examined here are its learning properties. The work has also addressed the interaction problem, providing a large amount of flexibility in interaction as described in the previous chapters. However, all interaction is driven by the agent's lacks of knowledge; whenever the agent does not know what to do next, it asks for help. Agent-initiated interaction is a poor approximation to the interaction between a human instructor and student, in which both parties initiate a variety of kinds of instruction events. One study has shown that teacher-initiated instruction dominates early in learning, with student initiation increasing later [Emihovich and Miller, 1988]. Instructo-Soar does not attempt to model this pattern of interaction.

There are five major behavioral properties of Instructo-Soar's procedural learning:

1. **Interactive instruction requests.** The agent asks for instruction as needed.
2. **Initial episodic learning.** The initial learning of a procedure is rote and episodic.
3. **Experiential, situated learning method.** The agent employs a situated "learning by (simulated) doing" method to produce explanations that lead to general learning.
4. **Final general learning.** The final form of a learned procedure is general and automatic, with each step directly conditional on features of the current situation.
5. **Monotonically increasing task knowledge.** As the agent receives more and more instruction, its task knowledge continues to increase; there is no structural limit to the amount of knowledge that can be learned.

Each of these properties will be described in greater detail later in the chapter, when the effect of Soar on each is analyzed. The next section examines the use of unified theories of cognition in developing theories of specific behavior, laying the groundwork for this analysis.

## 5.2 Cognitive modeling within unified theories of cognition

One of the primary problems to be faced when building a cognitive model is the *underconstraint* of the model by the behavioral data. There are a multitude of design decisions that must be made in building a model, and the data does not completely constrain those decisions. Thus, many models can account for the same data.

A recurring theme of research on Soar, and prior research of Allen Newell's, has been the useful constraint that a unified theory of cognition (UTC) offers in designing cognitive models [Newell, 1990; Newell, 1973]. A UTC, as embodied in the form of an architecture, posits a set of mechanisms

capable of supporting intelligent behavior. By constructing a model within a UTC, we restrict ourselves to designs consistent with the architecture's properties. This constraint is useful because it produces models based on principles that are globally motivated by a broad range of cognitive behavior and phenomena. Since models are based on a consistent set of underlying mechanisms, modeling within a UTC also facilitates the integration of models to cover a larger range of behavior.

The constraint provided by a UTC is most evident for tasks at low timescales. For these tasks, behavior is strongly determined by the architecture itself. This is because there is not enough time for more than a small number of applications of the architecture's primitive mechanisms. However, as the timescale increases, to the level of minutes or hours for task performance, behavior is determined more and more by an intelligent agent's ***knowledge***. This is Newell's intendedly rational band [Newell, 1990], in which the agent moves towards approximating a pure knowledge-level system. Learning from instruction falls into this band, because performance is dominated by knowledge, and takes on the order of minutes to hours. The large timescale allows the agent to retrieve, compose, and apply relevant knowledge to reach its goals. Because of the distance in timescales between knowledge level behavior and an architecture's primitive mechanisms, it is less evident how a UTC might constrain the design of models for behavior at these higher timescales. Thus, a key question to be addressed is, "What constraints, if any, does a UTC offer in designing a model of knowledge level behavior?"

The sections that follow present an analysis of the effect of a particular UTC, Soar, on a particular model of knowledge level behavior, Instructo-Soar. The analysis demonstrates a general methodology for analyzing the effect of a UTC, that utilizes ***architectural entailments*** derived from the architecture's basic mechanisms and properties. A UTC's entailments can powerfully guide the development of a model of knowledge level behavior. The analysis shows that each of the behavioral properties of the Instructo-Soar model mentioned above is affected by different architectural entailments of Soar.

## 5.3 Soar's architectural entailments

Architectural entailments are ramifications of the combined architectural components that define the UTC, that provide a level of description of an architecture above the level of the basic components. Examples of architectural entailments include whether or not the architecture's behavior is goal oriented, interrupt driven, based on a particular weak method, etc. Since they are at a higher level, entailments are more useful than the basic architectural properties for determining the architecture's effect on large timescale behavior.

Entailments provide constraint in designing knowledge level models because they determine which techniques, structures, etc., are most "natural" within the architecture. Models that are most *afforded* by the entailments are preferred. As Newell puts it: "There are many ways for Soar to...do any [intendedly rational] task. A natural way is one that involves little additional knowledge and little computational apparatus and indirection...One of the general arguments for the importance of getting the architecture right, even though it appears to recede as the time scale increases, is that doing what is natural within the architecture will tend to be the way humans actually do a task." [Newell, 1990, p. 416-17]

A UTC's entailments derive from its basic properties.

### 5.3.1 Basic properties of Soar

Following Newell [1990, p. 160], Soar can be described as having six basic properties:

1. **Problem spaces represent all tasks.** Soar follows the problem space computational model, with the basic computational structures of states and operators organized in problem spaces.

2. **Productions for long-term memory.** All long-term knowledge is stored in associative productions, including search control, operator application knowledge, and semantic and episodic knowledge.

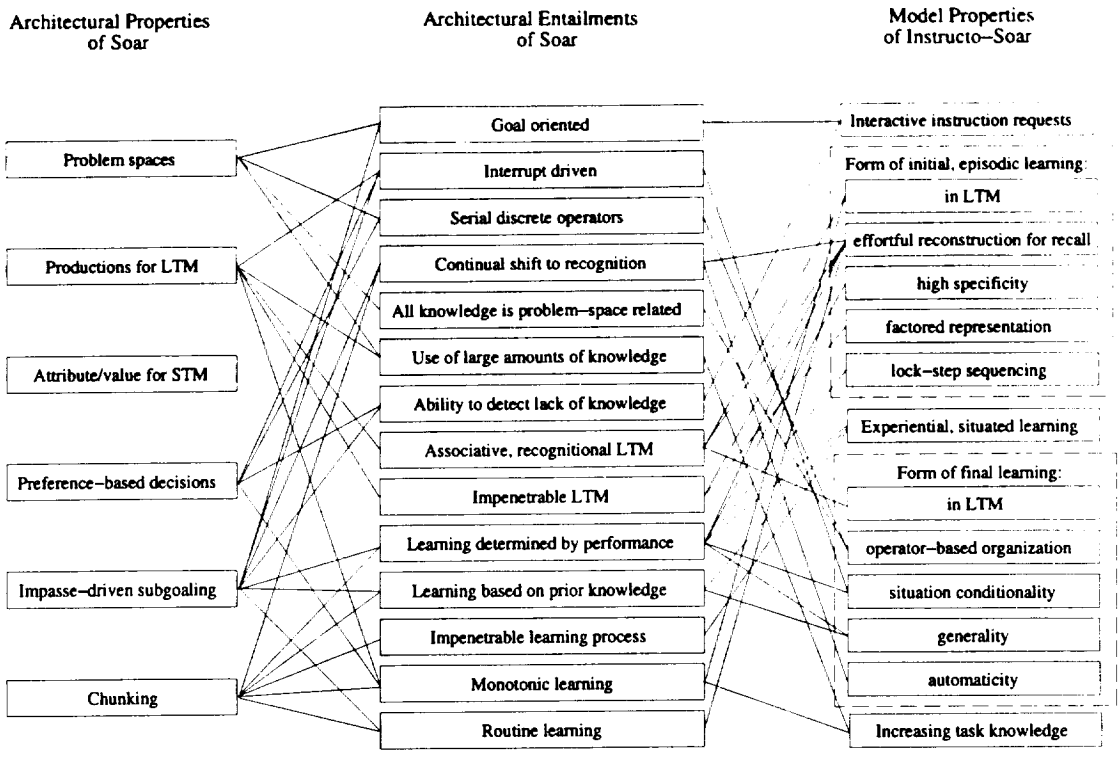



Figure 5.1: How the entailments of Soar arise and affect the Instructo-Soar model.

3. **Attribute/value representation.** All declarative structures in working memory are represented as objects with sets of attributes and associated values.

4. **Preference-based decision procedure.** Productions produce symbolic preferences for working memory elements, including problem spaces, states, operators, and substructures of these objects. The decision procedure interprets the preferences to select appropriate values for working memory.

5. **Impasse-based automatic subgoaling.** Subgoals are created automatically in response to the inability to make progress on a problem.

6. **Chunking based learning.** Chunking produces a new production whenever a result is created during processing in a subgoal. The new production, or chunk, summarizes the result of processing within the subgoal; in future executions of related tasks, chunks will fire, avoiding the subgoal.

### 5.3.2 Entailments of Soar

Starting from Newell's characterization of Soar [1990, pp. 227-30], a list of Soar's entailments has been generated based on its basic properties. The entailments are broken down into four categories: behavior, knowledge, long-term memory, and learning. The list is not intended to be exhaustive, but includes the key entailments that affect Instructo-Soar, as described in the next section. Each entailment derives from some combination of the basic properties of Soar, as indicated by numbers (from the list of Soar's properties above) in brackets after each entry in the list below. These dependencies are shown graphically by the first two columns of Figure 5.1.

- **Behavior.**

1. Goal oriented. Soar creates its own goals whenever it does not know how to proceed [1,5].
2. Interrupt driven. All productions are being matched continuously, and bring to bear whatever knowledge is appropriate to the current situation. The preference-based decision process allows knowledge to indicate that a more important course of action than the current course should be followed. Subgoals based on the old course of action are terminated when an alternative course of action is selected. [2,4,5].
3. Serial application of discrete, deliberate operators. A single operator is selected and applied at a time within each goal context. [1,4]
4. Continual shift to recognitional (impasse-free) performance. Whenever an impasse is resolved, chunks are learned that allow the impasse to be avoided in the future. These chunks implement recognitional performance – they simply recognize the situation and act appropriately, without requiring further deliberation (i.e. subgoals). [5,6]

- **Knowledge.**

5. All knowledge is related to problem-space structures: problem spaces, states, and operators. [1]
6. Use of an indefinitely large body of knowledge. Soar can use large amounts of knowledge because of the structuring of knowledge as individual productions, and the factoring of that knowledge into multiple problem spaces. [1,2]
7. Ability to detect a lack of knowledge. Impasses are detected architecturally, and indicate a lack of knowledge to make a decision. [4,5]

- **Long-term memory.**

8. Associative, recognitional long-term memory. Productions form an associative long-term memory. Knowledge is recalled only when appropriate retrieval cues appear in working memory. [2]
9. Impenetrable long-term memory. The productions in long-term memory cannot be examined by the agent, for instance by being tested by other productions. Thus, the agent is not fully aware of what is in its long-term memory; access is gained only through working memory retrieval cues. [2]

- **Learning.**

10. Learning determined by performance. The chunks that are learned are directly dependent on the processing that takes place in a subgoal to produce a result. In particular, the generality (transferability) of chunks is dependent on the generality of the problem solving in the subgoal. [5,6]
11. Learning based on prior knowledge. Since learning is dependent on performance, and performance is dependent on the agent's existing knowledge, it follows that what is learned depends critically on the agent's preexisting knowledge. [5,6]
12. Impenetrable learning process. The chunking process is impenetrable to the agent; it cannot learn to alter its architectural learning process.[2] [6]
13. Monotonic learning. Productions are never removed from long-term memory; once learned, chunks cannot be forgotten. This contrasts with work in machine learning involving deliberate forgetting mechanisms (e.g., [Markovitch and Scott, 1988; Salganicoff, 1993]). However, the effects of a chunk can be overridden by other chunks to recover from incorrect learning [Laird, 1988]. [2,4,6]

[2]However, since learning is dependent on performance and knowledge, chunking can form the basis for many different learning strategies (e.g., induction, analogical learning, abstraction, etc.). These strategies are essentially knowledge-level behaviors that are mediated by knowledge and can be learned.

**If** the operator is **resolve(<referent>)**, and
the discourse segment is **segment13**, and
**<referent> isa move-act**,
**then <referent>** is recognized as having **id i943**.

Figure 5.2: A recognitional chunk that encodes an episodic memory of one semantic feature of an instruction.

14. Routine learning. Chunking is part of the routine performance of the agent, rather than a special process that is deliberately invoked. The decision to learn is not under the agent's control (except indirectly as the agent decides when to return results from subgoals). [5,6]

## 5.4 Soar's impact on Instructo-Soar

In this section, each of the major behavioral properties of the Instructo-Soar model is examined in more detail. Each model property is affected by a variety of Soar's architectural entailments, revealing the constraining effect of developing the model within Soar.

### 5.4.1 Interactive instruction requests.

The agent asks for instruction whenever its available knowledge is insufficient to reach its goals. It can do this because it is goal oriented (entailment 1), and can detect its lack of knowledge (entailment 7).

### 5.4.2 Initial episodic learning.

As a side effect of natural language comprehension when reading the instructions for a new procedure, the agent learns a set of chunks which provides an episodic memory of the instructions. This memory has the following characteristics:

- **Long term memory.** The episode is encoded as a set of productions in Soar's long term memory (entailment 8). Thus, instruction sequences are remembered without an unrealistic load on short term memory, and instructions are not forgotten (entailment 13).

- **Recognitional memory; effortful reconstruction for recall.** The episode is encoded as a set of *recognition rules*, in which the features of the instruction to be recalled make up the *conditions*, rather than the actions, of each chunk. An example of an episodic recognition rule is shown in Figure 5.2. The rule tests a discourse segment symbol (**segment13**); a new discourse segment symbol is created each time the agent begins to receive instructions toward a new goal. The rule recognizes **isa move-act** as a semantic feature of an instruction that was given within **segment13**. Recognition is the basic form of long-term memory (entailment 8), and processing impasses results in recognitional learning (entailment 4); here as a side-effect of natural language comprehension (entailment 14). The chunks are recognitional because their conditions are dependent on the behavior in the subgoal they are produced from (entailment 10); and in this case, the behavior (language comprehension referent resolution) is dependent on the semantic features of the instruction.

  To recall parts of the instruction episode, the agent must deliberately reconstruct features of the episode, which are then recognized by these rules. This is because the agent cannot simply inspect the contents of long-term memory (entailment 9); it must be accessed by placing the appropriate recall cues in working memory.

- **High Specificity.** Each instruction is indexed by the exact goal it applies to (e.g., picking up the red block `blk-r`); similarly, the exact instruction given is remembered (e.g., moving to the yellow table `tbl-y`). The specificity results from learning based on performance (entailment 10); here, the performance (referent resolution) involves reference to specific features of the goal and the instruction. The specificity is required to avoid confusion between different referents associated with different goals (which are associated with specific discourse segments).

- **Factored representation.** There are separate recognition chunks for each semantic feature of each instruction, as illustrated by the chunk in Figure 5.2. The chunks are learned for the purpose of resolving future natural language referents, and this requires independent access to each feature (e.g., after reading "the red block", you may read "the red one" and need to recover the referent from only its color feature). Independent access requires a set of individual feature chunks because long-term memory cannot be examined (entailment 9); thus, the agent cannot simply memorize a monolithic semantic structure.

- **Lock-step sequencing.** Instructions are sequenced by being chained one to the next, instead of indexed directly by the situations they should apply in. Learning is performance based (entailment 10), and the performance of *reading* the instructions (which produces the episodic memory) does not depend on what situations the instructions will apply to during performance of the actual task.

### 5.4.3 Experiential, situated learning method.

By internally projecting instructed actions, the agent learns general rules that propose the actions directly when they will allow a goal to be reached. This experiential learning method is afforded because learning is based on performance (entailment 10) and is impacted by prior knowledge (entailment 11). Thus the "natural" way for a Soar system to learn about task performance is to perform the task itself (thereby making maximal use of the task domain knowledge it already has) – here, internally, in case the task cannot be completed. There are alternatives; for example, reasoning directly in the space of possible condition sets about the conditions under which each instruction might apply. However, any such method would require adding additional knowledge to Soar (about conditions, condition sets, how to search that space, etc.). In addition, it would be difficult to learn chunks with exactly the desired conditions, since the chunking process is impenetrable (entailment 12) – an agent cannot simply say "now I'll add a chunk to my memory with the following conditions and actions."

### 5.4.4 Final general learning.

The experiential, situated learning method allows Instructo-Soar to learn procedures in a general form that has the following characteristics:

- **Long term memory.** Rules implementing a procedure are encoded as productions (entailment 8).

- **Operator-based organization.** Actions and procedures are encoded as individual operators; the agent learns when to select these operators, how to achieve them, and how to recognize their achievement (entailments 3 and 6).

- **Situation conditionality.** Each operator proposal rule is directly conditional on relevant features of the current situation (state and goal being pursued) of the agent. This is because learning is determined by performance (entailment 10); here, the performance is an internal projection of the instructed operator from a particular state to achieve a particular goal. The implementation displays reactivity, since if the world state unexpectedly changes, rules proposing the next operator to perform will match or mismatch accordingly (entailment 2).

- **Generality.** The features in each proposal rule are only those *required* for successfully reaching the goal. This is because learning is determined by performance (entailment 10) – here, goal achievement – and prior knowledge (entailment 11) – here, knowledge of how to simulate the basic operators.

- **Automaticity.** The learned rules move the system to recognitional performance (entailment 4). No effort is required to recall instructed operators in the appropriate situation, as it is for the initial rote learning.

### 5.4.5 Monotonically increasing task knowledge

There is no structural limit on the amount of task knowledge the agent can acquire from instruction (entailment 6). No forgetting or symbol-level interference will occur; that is, the learning of new productions will not overwrite old ones (entailment 13). Interference may occur at the knowledge level, if productions bring conflicting knowledge to bear on the task. This indicates that there is an actual conflict in the agent's knowledge (e.g., perhaps the instructor has given instructions that conflict or overlap). In cases where conflicting operators are proposed, Instructo-Soar can learn to resolve the conflict by taking further instruction, as demonstrated in Section 4.3.

### 5.4.6 Analyzing Soar's effect

Figure 5.1 summarizes the relationship between the properties of Soar, the entailments they give rise to, and properties of the Instructo-Soar model. In essence, this figure factors the effects of each UTC property and entailment on the model. It gives a detailed answer to the question of how the use of the UTC impacted the model.[3]

The figure can be used to determine how each model property is influenced by the properties and entailments of the UTC. As an example, Figure 5.3 indicates which UTC properties influence the Instructo-Soar property that initial episodic learning requires an effortful reconstruction to be recalled. As the figure shows, a number of Soar's entailments and properties influence this model property, indicating that a large amount of constraint was provided by the architecture.

Going the other direction, the figure indicates what effect each property of the architecture has on the model. This can show which architectural properties have a large effect and which have no effect (provide no constraint). For instance, Figure 5.4 traces the effect of Soar's production based, recognitional form of long-term memory, which has a large impact on Instructo-Soar. On the other hand, the fact that Soar uses an attribute/value based representation has no effect. Such analyses can be useful in determining how the properties of a model might change if the architecture were changed in various ways.

Entailments provide an intermediate level of granularity that can be used to compare models within different unified theories. Where another architecture's entailments differ from Soar's, a figure like Figure 5.1 indicates the model properties that will be affected by the differing entailments. For instance, ACT* [Anderson, 1983] has a declarative long-term memory in addition to a production memory. This entails a fully penetrable, non-associative memory, that could be used to store instructions as they are read. From our analysis, the associative and impenetrable nature of Soar's long-term memory affects Instructo-Soar's form of initial, episodic learning in three ways: (1) the episode is in long-term memory, (2) it requires reconstruction for recall, and (3) it is encoded in multiple productions that factor the instruction features. In an ACT* based model of learning procedures from instruction, (1) the episode could still be in long-term memory, but now in declarative memory, (2) no reconstruction would be required for recall, but rather a search of declarative memory, and (3) the representation would not be factored into separate associative pieces; rather,

[3] The methodology used to construct this figure is somewhat similar to VanLehn's [1990] analysis of the assumptions in his SIERRA theory of procedural learning. A UTC's basic properties are like VanLehn's "assumptions" and its entailments are like his "theorems" (which are derived from the assumptions). A competitive argument can be made for the various model properties based on the entailments. However, VanLehn's goal is to understand the dependencies between assumptions in his theory and to argue for those assumptions, while the goal here is to understand the impact of a particular set of assumptions (a unified theory) on a resulting model.

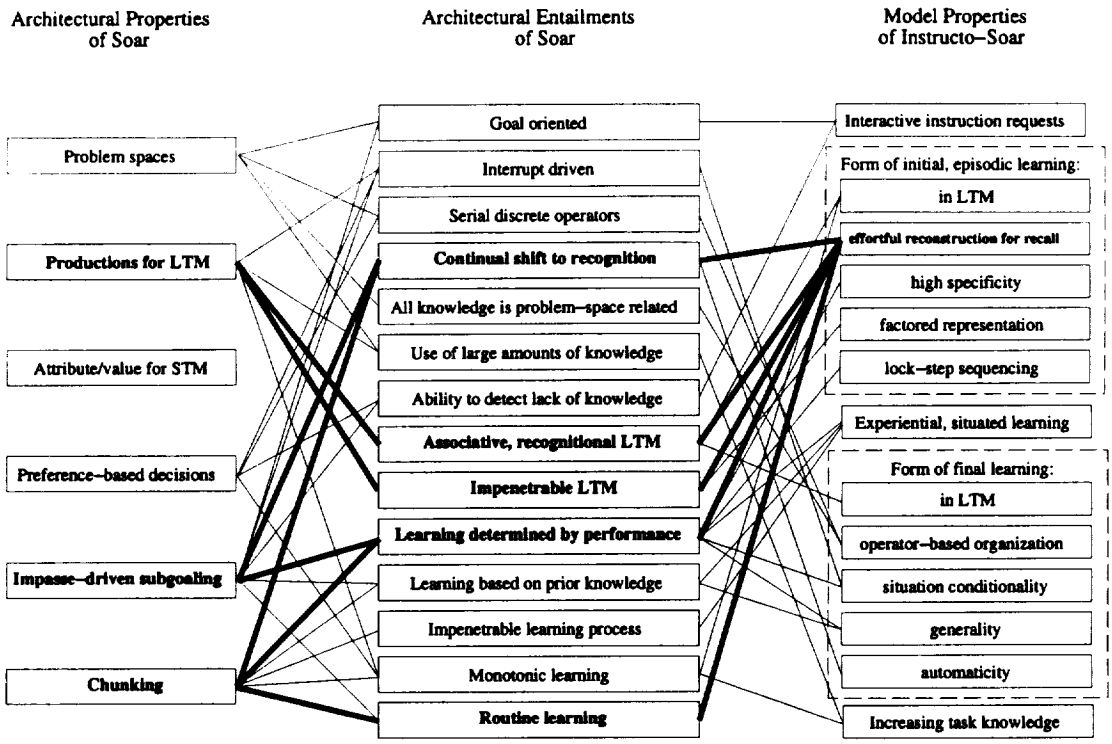



Figure 5.3: The properties and entailments of Soar that influence Instructo-Soar's property that initial episodic learning requires an effortful reconstruction to be recalled.

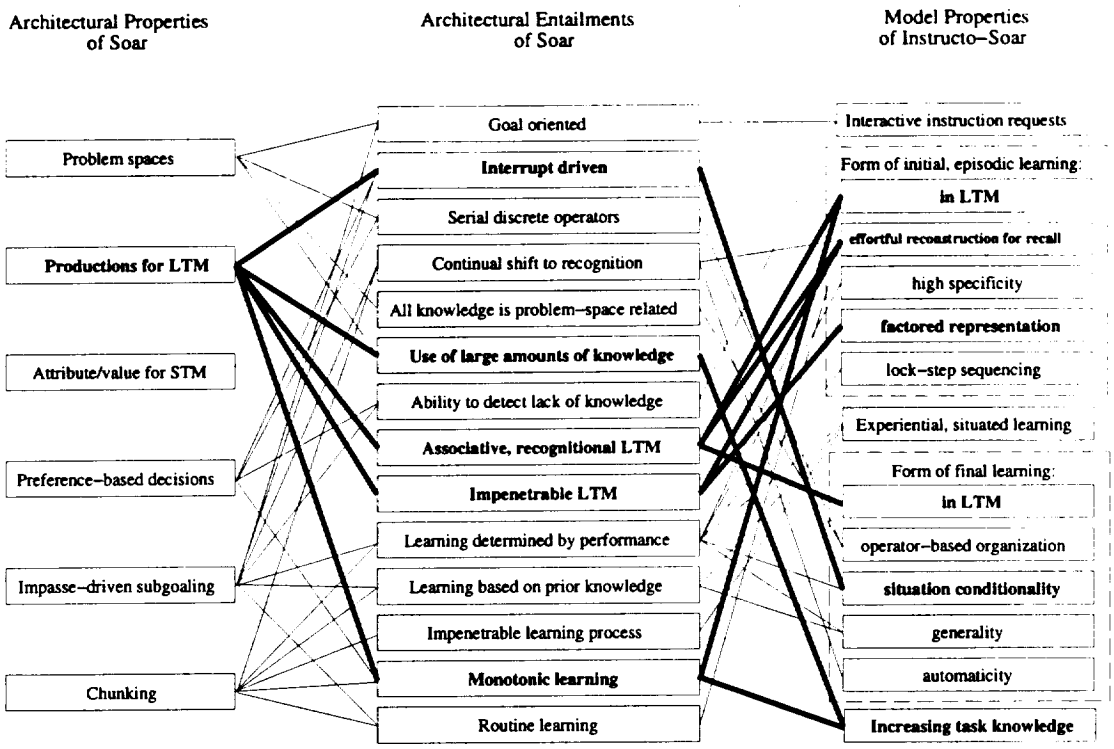



Figure 5.4: The effect of Soar's recognitional long-term memory on Instructo-Soar.

any feature could be retrieved from a declarative structure. These properties are indeed found in an ACT* model of procedural learning from instruction [Anderson, 1983]. This model will be contrasted further with Instructo-Soar after a discussion of Instructo-Soar's behavioral predictions.

## 5.5 Model predictions

Thus far, the Instructo-Soar model's properties have been described in terms dependent on the model's data structures and processes. However, in order to evaluate the model as a testable hypothesis, behavioral predictions must be described independent of the model and architecture. The model's predictions are determined by its behavioral properties. Determining predictions based on behavioral properties is a step that must be taken in any cognitive modeling effort, whether based on a unified theory or not.

Instructo-Soar makes a number of testable predictions. Some predictions are dependent on the recall strategy used to recall episodic memories of past instructions (described in Chapter 3). Except where otherwise noted, the "single recall" strategy will be assumed.

- Initial learning of a new procedure is rote and episodic. Thus:

  (1). The procedure can initially be performed only in situations very similar to the original situation it was taught in (low transfer). This is because of the high specificity of the rote memory.

  (2). Subjects will initially have difficulty performing the procedure starting from any state other than the original initial state (i.e. they can't start the procedure in the middle very well). This is because the rote memory is highly specific, and is sequenced in a lock-step fashion.

(3). Performance is initially slow. This is because the effortful reconstruction required to recall instructions from long-term memory takes time; the factored representation of the episode requires that each feature be reconstructed, and the results assembled, for a complete recall.

- General learning of a new procedure does not occur immediately, but takes multiple executions of the procedure. Thus:

(4). The amount of transfer – the ability to perform the procedure in novel situations – increases with the number of executions, until the maximum transfer level is reached. This is because learning occurs through an experiential process.

(5). The execution time decreases with the number of executions until the maximum transfer level is reached. This is because parts of the procedure that must be recalled from episodic memory require a time-consuming reconstruction process, whereas parts of the procedure that have been learned generally apply directly without a recall process. Thus, as more and more of the procedure is learned in general form, execution time drops. Empirically, this execution time decrease displays the power law of practice, as shown in Figure 3.9.

(6). Procedures taught hierarchically will be learned faster and more effectively (as described in Chapter 3). Faster learning of hierarchies is a consequence of (5) – when a procedure is broken into a hierarchy of subtasks, each has fewer substeps than the overall procedure, and learning and execution time are proportional to the number of substeps. More effective learning of hierarchies is a consequence of learning by a situated explanation process; the shorter explanations required for each subtask are less susceptible to errors in prior knowledge.

- After some number of executions, the procedure is fully generalized. At this point:

(7). Performance is automatic, occurring recognitionally; thus, performance is rapid. Subjects' knowledge of what action to perform next is retrieved directly from long-term memory based on current situational features.

(8). Subjects will exhibit the *Einstellung* effect [Luchins, 1942]; that is, once a method is learned for a particular task, that method will continue to be used even if a more efficient method is available. This is because after general learning, performance is automatic and recognitional.

(9). Subjects will be able to execute the procedure equally well starting from any of the intermediate states along the path of the instructed action sequence. This is because the steps are operators indexed by current state conditions, rather than sequenced in a lock-step fashion, or stored as a monolithic plan.

- General learning is the result of an experiential, situated learning method. The method involves a deliberate recall and explanation process, applied to each individual instruction, using prior domain knowledge (e.g., knowledge of basic operators' effects). Thus:

(10). Self-explanation [Chi *et al.*, 1989] by subjects improves transfer. Subjects who do not attempt such explanations will achieve only low transfer learning (rote learning); subjects who successfully self-explain will achieve high transfer, general learning.

(11). Self-explanation improves performance time, because performance time is lower after general learning ((6), above).

(12). (Alternative recall strategies:) Because recall and explanation of instructions is an effortful, time-consuming process, when pressured to act quickly subjects may not have time to recall and explain an entire instruction sequence at once. Thus, under more time pressure, the number of executions of a procedure to achieve high transfer learning will increase (moving towards the single recall strategy). Under less time pressure, or when

encouraged to introspect about their behavior, subjects will achieve high transfer learning more quickly (moving towards the introspective recall strategy). However, although it models different recall strategies, Instructo-Soar does not explicitly model subjects' switching between them.

(13). Because of the explanation-based nature of the learning process, the amount and quality of transfer achieved depends on the amount and quality of prior domain knowledge of the subject – subjects with less domain knowledge will achieve less transfer.

(14). Unsituated instruction – instruction that does not clearly apply to a particular situation (either the current situation or a specified hypothetical one) – will be more difficult for subjects to learn from than situated instruction. This is because of the situated nature of the learning process.

- Task knowledge monotonically increases with learning from instruction. There is no structural limit on the amount that can be learned. Thus:

(15). Once learned, non-conflicting task knowledge will not be forgotten by subjects either because of disuse or other unrelated learning. (As all knowledge has associated recall cues, however, recall of a piece of knowledge may fail because the proper cues cannot be generated.)

Figure 5.5 shows these predictions and indicates the behavioral properties each arises from.

## 5.6 Comparison to human performance

As mentioned previously, there is little specific data on human learning and transfer during the course of tutorial instruction. The kind of data needed to test Instructo-Soar's predictions would include detailed timing data within and across executions of a new task, and carefully collected transfer data between executions; experiments would also have to control for subjects' prior knowledge. No detailed fit of Instructo-Soar to specific data has been attempted.

However, Instructo-Soar's predictions and properties appear to be broadly consistent with what is known about human performance. Four general human learning effects that are consistent with the Instructo-Soar model are the role of situatedness, the effect of prior knowledge, the effect of self-explanation, and the ability to transfer learned knowledge. In the rest of this section, each of these will be discussed in turn.

### 5.6.1 Situated learning

There is a growing recognition that situated learning – learning that results from performing or participating in performance of tasks in actual task situations – is highly effective [Vera *et al.*, 1993; Lave and Wenger, 1991; Brown *et al.*, 1989; Emihovich and Miller, 1988]. Knowledge is only useful to the extent that it can be applied correctly in particular situations [Singley and Anderson, 1989; Chi *et al.*, 1989]; and it appears that people learn to apply knowledge within situations by performing tasks in those situations. This idea underlies the increase in training through computer simulated practice (e.g., pilot training) [Lintern, 1991].

Actually performing procedural instructions produces different memory effects than simply memorizing them as text [Kintsch, 1986]. In fact, students seem to *prefer* instructions that can be performed to achieve a particular task over more general, unsituated instruction. As Carroll [1984] observes, students "don't appreciate overviews, reviews, and previews; they want to do things." This preference is so strong that "they are apt to plunge into a procedure as soon as it is mentioned or will try to execute purely expository descriptions" [Carroll, 1984, p. 125]. Carroll's studies have shown that training that leads students through execution of actual tasks is more effective than other kinds of training.

Instructo-Soar predicts the effectiveness of situated learning because its central learning process is situated explanation. Situated explanation requires that the agent understand what goal

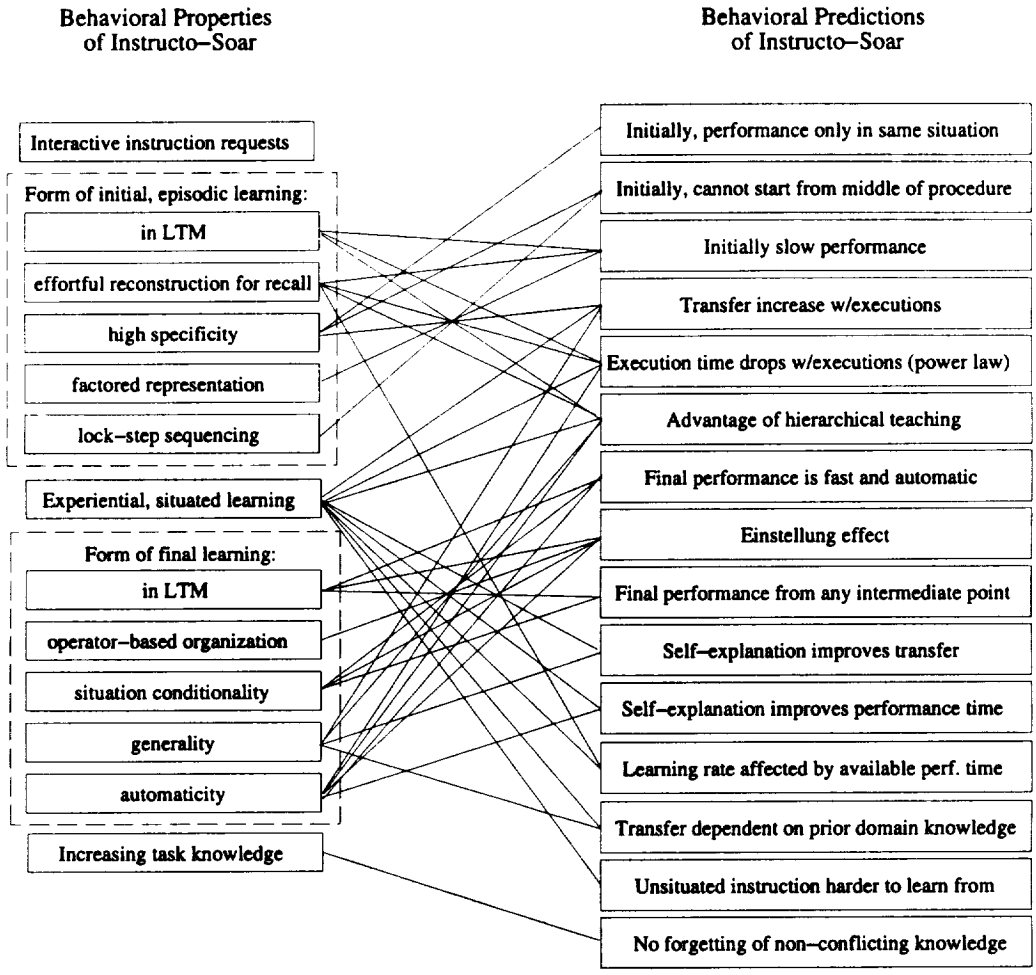



Figure 5.5: Instructo-Soar's behavioral predictions, as they arise from its behavioral properties.

and state (situation) an instruction applies to, before it can learn generally from the instruction. Instructo-Soar predicts an advantage for hierarchical instruction because it breaks a procedure into smaller intermediate goals that are easier to explain. Hierarchical instruction that provides information about intermediate goals has been shown to be more effective than purely linear step-by-step instruction [Smith and Goodman, 1984; Konoske and Ellis, 1986]. Goal information also helps students interpret commands correctly [Dixon, 1987]; this is a part of the mapping problem that is not modeled by Instructo-Soar.

Instructo-Soar uses a situated learning process even for instructions that apply to a situation different from the current situation. To learn from explicitly situated instructions like conditionals, it creates an appropriate internal situation and simulates performance of the instruction. This is similar to other models of interpreting conditionals, such as Johnson-Laird's [1986].

### 5.6.2 Prior knowledge

Psychological studies have shown that subjects' prior domain knowledge affects their ability to both interpret and learn from instructions. Instructo-Soar does not focus on interpretation (the mapping problem). On the other hand, general learning in Instructo-Soar depends strongly on the agent's prior domain knowledge. If domain knowledge is lacking or of low quality, the learning will degrade, falling back on naive heuristics or simply rote learning. This is a good match to observed human behavior.

*Instruction interpretation:* Instructo-Soar's simple approach to the mapping problem does not model these effects, but they are mentioned for completeness. Dixon *et al.* [1988] gave subjects instructions with key actions stated either explicitly or implicitly, as a part of another step. They found that readers with a high amount of background knowledge were less effected by the implicit form than those with low background knowledge. Mohammed and Swales [1984] tested subjects who were either native or non-native speakers, with either a science or non-science background, in following instructions to set the time and alarm on a digital clock. Non-native speakers with a science background performed better than native speakers without a science background. In both of these studies, subjects with prior knowledge may have made up for incomplete comprehension of the instructions by inferring the missing steps, whereas low knowledge subjects could not do this [Bovair and Kieras, 1991].

*Learning and transfer:* Prior domain knowledge has been shown to improve students' instructional learning and transfer. Judd [1908] had two groups of boys throw darts at underwater targets. One group, taught about refraction beforehand, outperformed the other after the basic mechanics of the task were mastered. More recently, Kieras and Bovair [1984] taught subjects procedures for operating an unknown device (a Star-Trek like control panel). Subjects given prior instruction about the device's inner workings (a domain theory for the device) learned the procedures more effectively, in terms of speed and accuracy of performance, and retention of the procedures. Churchill and Young [1991] built a simple production based model of these results, that produces rote, strictly ordered rules from instruction without prior device knowledge, and more general, situated rules when it has prior device knowledge. In Instructo-Soar, the generality, accuracy, and execution time of knowledge that is learned are all strongly dependent on the agent's prior knowledge.

### 5.6.3 Self-explanation

Chi *et al.* [1989] took protocols of subjects studying worked out examples of physics problems, and then measured the subjects' performance on a test containing similar problems. They found that subjects who performed well on the test problems had produced significantly more self-explanations while studying the examples. These self-explanations involved a deliberate, conscious effort to justify the steps in the example, by explicating consequences and preconditions of actions [Chi and VanLehn, 1991]. Students formed explanations of individual steps in each example, rather than explaining multiple steps [VanLehn *et al.*, 1992]. Poor students did not self-explain but rather processed the examples in a verbatim fashion.

During problem solving, good solvers (those who self-explained the examples) referred to examples sparingly, and in a targeted fashion, looking for specific information (e.g., a specific equation). Poor solvers, on the other hand, used the examples extensively as syntactic templates. Based on surface features, they would try (often inappropriately) to apply actions taken in an example problem to their current problem. This rote use of the examples did not work very well, and resulted in low transfer. VanLehn *et al.* [1992] present a detailed model of these effects, called CASCADE, that is able to successfully reproduce the performance of individual subjects.

These results are relevant because Instructo-Soar employs a self-explanation approach to learning from tutorial instruction. Instructions are explained in a line-by-line manner to "justify" each instructed action, based on an internal projection that elucidates its effects. Although example problems are a somewhat different form of instruction than tutorial instruction, Instructo-Soar's predictions are similar to the self-explanation findings. If Instructo-Soar does not attempt to explain instructions, its learning will degrade to rote learning. This rotely learned knowledge could be applied analogically to future problems based on surface similarity, just as the poor students in the self-explanation studies used the examples (although this is not implemented in Instructo-Soar). However, only by successfully self-explaining an instruction is high transfer, general learning achieved. This is resonant with Chi and VanLehn's [1991] conclusion:

> Our data suggest that students can learn, with understanding, from a single or a few examples...However, only those students who provide adequate explanations during studying are able to see the degree to which they can generalize...Our results suggest that unless the student self-explains from examples, there is little opportunity for transfer [Chi and VanLehn, 1991, p. 101].

Instructo-Soar's self-explanations take the form of mental simulations of instructed actions. There is ample evidence that this type of "mental practice" results in useful learning for human subjects [Annett, 1991]. Feltz and Landers [1983] and Johnson [1982] found evidence that mental practice causes rehearsal of cognitive processes (as in Instructo-Soar), not only low-level motor responses.

### 5.6.4 Transfer

Other than the self-explanation effect, two other general transfer effects are found in Instructo-Soar's behavior. First, over multiple executions, the execution time of a procedure being learned follows the power law of practice [Rosenbloom and Newell, 1986]. This is a within-task positive transfer. Second, once learned generally, an inefficient procedure will exhibit the Einstellung effect; it will be difficult to learn a more efficient procedure. This is negative transfer.

Instructo-Soar, and Soar in general, is one of a class of theories that explain transfer by a modernized version of Thorndike's [1903] transfer of identical elements theory [Singley and Anderson, 1989; Gray and Orasanu, 1987]. The shortcoming of Thorndike's original formulation is that the identical elements were highly specific stimulus-response rules. In the modernized versions, these overly specific rules have been replaced by more general associative constructs, such as productions. Transfer occurs when productions that are used during one task also apply to another task. Since productions can be very general, transfer can occur across disparate tasks. However, the majority of productions for any particular task will be quite specific. Likewise, most results on transfer indicate that it is very narrow [Singley and Anderson, 1989; Bovair and Kieras, 1991]. For instance, Foltz *et al.* [1988] found that simply changing the name of a procedure from *delete* to *erase* caused a failure to transfer. This is consistent with Instructo-Soar, where general chunks learned for particular procedures are always dependent on the procedure's name and other situational features.

## 5.7 Other theories

As a cognitive model, Instructo-Soar can be compared with other cognitive models of procedural learning from instruction. Most recent models fall into two major classes: models based on ACT*, and case-based models.

### 5.7.1 ACT*-based theories

Many recent models of procedural learning are based on Anderson's ACT* model [Anderson, 1983; Anderson, 1987; Singley and Anderson, 1989]. Anderson's model employs a three stage process. First, instructions are converted into a declarative form in long-term memory. Second, the declarative memories guide task execution via weak methods, and through this process, new task-specific rules for the procedure are learned. Third, these rules are tuned and strengthened through future executions. The initial, declarative encoding of instructions is not use specific, but can be accessed by any process. The final, proceduralized knowledge, on the other hand, is applicable only in a specific set of situations. Transfer occurs when productions, as identical elements, apply across multiple tasks.

This model has been applied to a large amount of both aggregate and individual human data. It is intended to model procedural learning from a variety of types of input, for complex tasks like physics problems. Instruction about these tasks often includes declarative information, like the fact that $F = m \cdot a$, given in an unsituated way, rather than at a particular point in a particular problem. This kind of unsituated information is outside of the scope of tutorial instruction as defined in this thesis. To make use of such information when performing tasks, ACT* uses weak methods to decide what to do, and uses the instructed information to guide the weak method in whatever way it can. In contrast, Instructo-Soar's instructions are always either implicitly or explicitly situated; that is, they are always given within the context of a particular problem. Thus, the "weak method" used during task execution is simply to perform whatever action is indicated by the appropriate instruction.

These caveats aside, Instructo-Soar has many similarities to the ACT*-based theories. As in ACT*, initial reading of instructions produces long-term memories, but they are not fully operationalized; execution of tasks through deliberate recall of instructed information fosters learning of general productions that encode the instructed knowledge; and final learning is use specific, producing automatized behavior by directly applying to particular situations. Transfer of learning occurs when productions that were learned on an earlier task are applied to a later task.

However, there are differences. One is the form of initial learning. In Instructo-Soar, initial learning is episodic in nature: highly specific and recognitional. The encoding of an instruction exists declaratively within short term memory only for a short time, just after being read. To recall past instructions, the system must use an effortful, reconstructive process. In ACT*-based theories, instructions are encoded into a declarative long-term memory, that can be accessed without a reconstructive process.

A second difference is the source of the power law of practice. In ACT*, during the first execution of a task, the knowledge needed to perform that task is fully proceduralized (encoded into general rules). The power law arises because of an additional strengthening mechanism applied during later executions that makes the rules apply faster. In Instructo-Soar, knowledge is proceduralized across multiple executions of the task, rather than being fully proceduralized during the first execution. The power law arises from this incremental proceduralization.

A third difference is Instructo-Soar's emphasis on the use of prior knowledge and self-explanation to produce general learning. Models built within the ACT* theory have focused primarily on situations in which there is little prior domain knowledge, and initial performance of tasks requires applying weak methods.

Other ACT* type theories include models by Lewis *et al.* [1989] and Bovair and Kieras [Bovair, 1991; Bovair and Kieras, 1991; Kieras and Bovair, 1986]. Lewis *et al.* [1989] describe a system, built in Soar, that reads instructions to perform a subject's part in simple psychological experiments. The instructions are steps in a procedure, that are read as a "recipe" in a non-interactive fashion. Reading produces declarative structures, and the system learns the procedure by interpreting these structures to execute the task. However, the procedure is simple enough that learning it does not demand any generalization of the instructions. Bovair [1991] and Bovair and Kieras [1991,1986] present learning results for subjects learning to operate an unknown device, and describe an ACT*-like model of the results. However, the instructions in these experiments were non-interactive; subjects were required to memorize all of the instructions for a procedure before attempting to

perform it. Learning under these conditions could be very different from interactive instruction [Bovair and Kieras, 1991]. Indeed, one of Bovair and Kieras' key results concerned a drop in reading times for instructions read after they had already been mastered by a student; in interactive instruction, however, a student typically does not ever *receive* an instruction again once it has been mastered. In addition, Bovair and Kieras' subjects (and thus their model) had no prior domain knowledge. The model makes no use of prior knowledge or explanation; rather, learned rules are generalized by heuristically applying syntactic generalization operations to them.

### 5.7.2 Case-based theories

In addition to ACT*-based theories, some recent research has attempted to produce case-based theories of learning from instruction-like input. Recent case-based theories of instructional learning include those of Alterman *et al.* [1991], and Redmond [1992]. Unfortunately, these are difficult to compare to Instructo-Soar directly, as they each attack different instructional tasks.

Alterman *et al.*'s [1991] FLOBN focuses on the task of altering an old procedure for use in a new situation, through instruction. FLOBN begins with the knowledge of a procedure for using a pay phone, and attempts to use it to make a call with an Airfone. When the procedure fails, the system uses the instructions for the Airfone to alter its pay phone procedure, learning a new Airfone procedure from the process. In contrast, Instructo-Soar focuses on the task of learning completely new procedures and extensions of incomplete procedures from instruction. FLOBN's task involves altering past domain knowledge that has been incorrectly applied; Instructo-Soar's task involves extending past domain knowledge that is incomplete.

Redmond's [1992] CELIA is a learning apprentice system that learns by observing expert actions. The system must infer the goals of various actions it observes, using a plan recognition process, and learn to achieve these goals as the instructor does. CELIA targets learning from observation of action sequences; in contrast, Instructo-Soar targets learning from interactive instruction, and focuses on supporting a broad range of interactive flexibility. Because of the nature of interactive instruction, Instructo-Soar always knows what goals it is pursuing (although it may not know their termination conditions), and thus does have to deal with inferring the instructor's goals from observations. On the other hand, in interactive instruction, steps can be skipped or commands can be unknown to the agent altogether; CELIA does not need to deal with these possibilities, because observed actions always occur in order, without skipping steps, and without unknown steps.

Although these models apply to different instructional tasks, they do have some overall similarities. As described in Chapter 5, Instructo-Soar's episodic memory has a case-based flavor, although the form of its final learning is more fine-grained, including separate, situated knowledge for each individual step in a procedure. In addition, all three of the systems use an explanation-based approach to learning, with inductive processes to fill in needed knowledge.

## 5.8 Conclusion

This chapter has examined Instructo-Soar from a cognitive perspective, and elucidated its behavioral properties and predictions. Instructo-Soar is a nascent cognitive model. Certain aspects of its behavior clearly do not model human performance (such as instruction initiated only by the student), and its performance has not been matched to specific human data. Nevertheless, it is a real cognitive model, in that it makes a number of testable behavioral predictions and meshes well with general cognitive effects in human procedural learning and transfer.

In deriving Instructo-Soar's properties, the chapter presented an analysis of how being developed within Soar affected Instructo-Soar. Proponents of unified theories of cognition make the claim that working within a unified theory provides constraint for developing microtheories of specific behavior. The analysis technique used in this chapter represents a general methodology for deriving that constraint. Rather than only describing the UTC's impact in a broad, vague way, the analysis factors the effects of individual properties and entailments of the UTC on the microtheory. This type of fine grained analysis is a powerful tool for discovering which aspects of a unified theory are crucial to a model developed within it, and which are unimportant. The analysis of Instructo-Soar

indicates that development within a unified theory did indeed provide a large amount of constraint on the model's design.

# Chapter 6

# Results, Limitations, and Future Work

This chapter reviews and analyzes the results achieved by Instructo-Soar and the theory of learning from tutorial instruction that it embodies. It begins by examining how Instructo-Soar measures up to the requirements of a complete tutorable agent laid out in Chapter 1, and summarizing the capabilities that the agent demonstrates. Instructo-Soar performs a number of distinct instructional tasks, but all are based on the situated explanation method applied to a PSCM-based agent. The agent's capabilities are demonstrated in an extended instructional scenario that is summarized here and described in detail in Appendix A.

Following the discussion of results, the chapter presents a detailed analysis of the limitations of the current theory and implemented agent, and (conversely) the opportunities for further research. These arise from limitations inherent in the tutorial instruction paradigm, limits of the agent's computational model and non-instructional capabilities, and incomplete solutions to each of the key instructional problems: the mapping problem, the interaction problem, and the transfer problem. Each of these is discussed in detail. Finally, to focus the discussion of future work, the chapter makes a specific proposal for the next generation of instructable agent, that addresses an interrelated subset of the present limitations.

## 6.1 Results

The ultimate goal of this line of research is to produce agents that can be instructed naturally and flexibly, as humans can be. The first chapter developed a set of requirements for a complete tutorable agent, meant to represent this goal. Instructo-Soar does not meet all of these requirements; rather, from the complete requirements a subset of those relating to the interaction and transfer problems were selected as targets for this research. Table 6.1 (a repeat of Table 4.1) shows the targeted requirements of a tutorable agent, as they are met by Instructo-Soar.

In meeting the requirements of Table 6.1, Instructo-Soar moves closer to the goal of a complete tutorable agent than previous learning apprentice systems. The requirements that have been met include three crucial evaluation criteria:

1. The agent is able to learn all of the types of knowledge it uses in task performance from instruction.

2. The agent supports full flexibility of knowledge content for instructions commanding actions. That is, the instructor can give a command for a task at any point during instruction, whether or not the agent knows the task or how to perform it in the current situation.

3. The agent can learn from implicitly situated instructions, and from each type of explicitly situated instructions.

In meeting these requirements, Instructo-Soar displays some nineteen distinct instructional capabilities. These capabilities are listed in Table 6.2. An important point of this work is that this

| | Requirement | | Met? | Where described |
|---|---|---|---|---|
| 1. | General learning from specific cases | | √ | Chapters 3-4 |
| 2. | Fast learning (each task instructed only once) | | √ | Section 3.4 |
| 3. | Maximal use of prior knowledge | | √ | Chapters 3-4 |
| 4. | Incremental learning | | √ | Chapters 3-4 |
| 5. | Ability to learn all types of knowledge used in task performance | | √ | Chapters 3-4 |
| | a. state inference | | √ | Section 4.1.1 |
| | b. operator proposal | | √ | Sections 3.4, 4.1-2 |
| | c. operator preferential: | rejection | √ | Section 4.1.4 |
| | | comparison | √ | Section 4.3 |
| | d. operator effects | | √ | Sections 4.2.2-3 |
| | e. operator termination | | √ | Section 3.4 |
| 6. | Agent-initiated instruction | | √ | Chapters 3-4 |
| 7. | Flexibility of knowledge content | | √ | Chapters 3-4 |
| | a. full flexibility for action commands | | √ | Chapter 3 |
| | b. others as appropriate | | √ | Chapter 4 |
| 8. | Situation flexibility | | √ | Chapters 3-4 |
| | a. implicitly situated | | √ | Chapter 3 |
| | b. explicitly situated: | hypothetical state | √ | Section 4.1 |
| | | hypothetical goal | √ | Section 4.2 |

Table 6.1: The target requirements for a tutorable agent that are met by Instructo-Soar.

myriad of instructional behavior does not require nineteen different learning techniques, but arises as a single general technique, situated explanation in a PSCM-based agent, is applied across the broad range of situations required by the comprehensive learning task of tutorial instruction.

Although many of the requirements for a complete tutorable agent have been met, others have not been. Instructo-Soar's performance on the complete set of requirements for a tutorable agent is shown in Table 6.3. As the table indicates, Instructo-Soar has met or partially met seven of the eleven criteria. Those that are not fully achieved indicate areas for further work, to be discussed later in the chapter.

The next section turns to empirical evaluation of Instructo-Soar, and presents the results of an instruction scenario demonstrating its capabilities.

### 6.1.1 Empirical evaluation

Most empirical evaluations of machine learning systems take one of four forms, each appropriate for addressing different evaluation questions:

A. *Comparison to other systems.* This technique is useful for evaluating how overall performance compares to the state-of-the-art. It can be used when there are other systems available that do the same learning task. For example, a concept learner's performance might be compared to ID3's.

B. *Comparison to an altered version of the same system.* Typically, the system is compared to a version of itself without some key component (so called "lesion studies"), to evaluate the impact of that component on the system's overall performance. This method is appropriate, for instance, when a new component has been added to a standard method.

C. *Measuring performance on a systematically generated series of problems.* This technique evaluates how the learning method is affected by single dimensions of the input. For example, a concept learner might be tested with more and more noisy training data. This method can be used when the input for the learning problem has dimensions that can be systematically altered.

| | Instructional capability | Example | Described |
|---|---|---|---|
| 1. | Learning completely new procedures | **pick up** | Sec. 3.4 |
| 2. | Taking instruction to alter/verify newly inferred knowledge (e.g. new operator's termination conditions) | **pick up**'s termination conditions | Sec. 3.4.2 |
| 3. | Extending a known procedure to apply in a new situation | put down before picking up | Sec. 3.5 |
| 4. | Hierarchical instruction: instructions for a new procedure embedded in other instruction | teaching **grasp** within **pick up** | Sec. 3.4 |
| 5. | Prohibited actions | "Never grasp red blocks." | Sec. 4.1.4 |
| 6. | Explanations of prohibited actions | "Red blocks are explosive." | Sec. 4.1.4 |
| 7. | Recovery from indirect achievement of a prohibited state | closing hand around explosive block | Sec. 4.1.4 |
| 8. | Inferences from simple statements – specific | "The grey block is metal." | Sec. 4.1.1 |
| 9. | Inferences from simple statements – generic | "White magnets are powered." | Sec. 4.1.1 |
| 10. | Inferences from conditionals | "if *condition* and *condition* then *concluded state feature*" | Sec. 4.1.1 |
| 11. | Operators to perform for hypothetical goals | "To turn on the light, push the red button." | Sec. 4.2.1 |
| 12. | Operators to perform in hypothetical states | "If the light is bright then dim the light." | Sec. 4.1.2 |
| 13. | Contingency operators to perform in related situations | "If block is metal then grasp magnet" | Sec. 4.1.3 |
| 14. | Explanations of operators for hypothetical states | "Dim lights use little electricity." | Sec. 4.1.2 |
| 15. | Learning perceivable operator effects | pushing the button dims the light | Sec. 4.2.2 |
| 16. | Learning non-perceivable operator effects, and associated inferences to recognize the effects | moving down over a metal block when holding a powered magnet | Sec. 4.2.3 |
| 17. | Learning procedures in spite of an incomplete domain theory | No frame axioms experiment | Sec. 3.6 |
| 18. | Dealing with operator conflicts: learning which operator to prefer | two ways to grasp a small metal block | Sec. 4.3 |
| 19. | Dealing with operator conflicts: learning choices are indifferent | two ways to grasp a small metal block | Sec. 4.3 |

Table 6.2: A list of instructional capabilities demonstrated by Instructo-Soar.

| Problem | Requirement | Met by Instructo-Soar? |
|---|---|---|
| Mapping | Mapping of all representable information | no (as needed to show other capabilities) |
| Interaction | Flexible initiation of instruction | no (only impasse-driven agent-initiated) |
| | Flexibility of instructional knowledge content | partial (full flexibility for action commands) |
| | Situation flexibility | partial (implicitly and explicitly situated instructions, but no multiple utterance hypotheticals) |
| Transfer | General learning from specific cases | yes (via situated explanation) |
| | Fast learning | yes (new procedures only taught once) |
| | Maximal use of prior knowledge | yes |
| | Incremental learning | yes |
| | Ability to learn all types of knowledge used in task performance | yes (all PSCM knowledge types) |
| | Ability to deal with incorrect knowledge | no (only extending incomplete knowledge) |
| | Learning from instruction coexisting with learning from other sources | no (not demonstrated) |

Table 6.3: Instructo-Soar's performance on the requirements for a complete tutorable agent.

D. *Measuring performance on known hard problems.* Known hard problems provide an evaluation of overall performance under extreme conditions. For instance, concept learners' performance is often measured on standard, difficult datasets. This method can be used when such benchmark problems are known and available.

Typically, there is a single kind of learning to be measured, and the goal is to show more powerful learning of that kind along some metric (e.g., concept accuracy per training example). Often, these evaluation methods are combined; for example, Shavlik *et al.* [1991] use (A), (C), and (D) to extensively compare symbolic and neural concept learning algorithms.

These empirical evaluation techniques are difficult to apply in great depth to learning from tutorial instruction for two reasons. First, whereas most machine learning efforts concentrate in *depth* on a particular type of learning, tutorial instruction requires *breadth* of a wide range of learning from a wide range of instructional interactions. Whereas depth can be measured by quantitative performance, breadth is measured by qualitative coverage. Second, tutorial instruction has not been extensively studied in machine learning, so there is not a battery of standard systems and problems available. Nonetheless, evaluation techniques (B), (C), and (D) have been applied to Instructo-Soar to address specific evaluation questions:

A. *Comparison to other systems:* This evaluation technique could not be applied because there are not other tutorial instruction systems available for comparison.

B. *Comparison to altered version:* A "lesion study" was performed to illustrate the effect of prior knowledge on the agent's performance, as described in Chapter 3. Without prior knowledge, the agent is unable to explain instructions, and must resort to inductive methods combined with instructor verification of the induced knowledge. This increases the amount of instruction required and can reduce learning quality. Chapter 3 also compares the effects of versions of the agent using different instruction recall strategies (Figure 3.14). Other than prior knowledge and recall strategies, it would be difficult to alter any single component of the agent and still produce comparable performance.

C. *Performance on systematically varied input:* The input to the agent – the instructions – can be altered in a number of ways to produce comparable learning. An empirical alteration that is described in Chapter 3 is the length of the instruction sequence for a procedure. As the graphs of Figures 3.9 and 3.10 show, Instructo-Soar's execution time for an instructed procedure varies with the length of the instruction sequence, and drops with number of executions according to a power law function until the procedure has been learned in general form. A second alteration of instruction is the teaching of a procedure through hierarchical subtasks rather than as a flat instruction sequence. Based on the power law result, it was predicted that hierarchical instruction would have a learning time advantage over flat instruction. Figure 3.13 shows empirical results confirming this prediction. A third alteration of instructional input is that a large number of instruction orderings can be used to teach a particular procedure, since the instructor may skip steps or organize the procedure into arbitrary subtasks during instruction. Instructo-Soar is able to handle these possibilities, as described in Chapter 3. Rather than an experimental analysis, a mathematical analysis has been performed to determine what range of flexibility these abilities provides to an instructor. The analysis, presented in Appendix B, finds that number of instruction sequences that can be used to teach a given procedure is very large, growing exponentially with the number primitive steps in the procedure.

D. *Performance on a known hard problem:* Since learning from tutorial instruction has not been extensively studied in machine learning, there are no standard, difficult problems. To create such a problem, the key evaluation criteria for learning from tutorial instruction mentioned at the beginning of this chapter were combined to produce a single scenario testing each of them. Instructo-Soar's performance on this scenario is presented in detail in Appendix A, and summarized briefly below.

The instruction scenario created to evaluate Instructo-Soar's breadth of performance requires its full range of learning and interaction capabilities. The scenario demonstrates eighteen of Instructo-Soar's nineteen instructional capabilities from Table 6.2 (it does not include learning indifference in selecting between two operators). It contains 100 sentences that teach a hierarchy of new operators, organized as shown in Figure 6.1, and a variety of knowledge about controlling the light and using the electromagnet. Instructo-Soar begins with 2,579 productions, and learns 4,730 chunks during the example, for a final total of 7,309 productions. The example takes 13,820 decision cycles (each decision cycle, a Soar agent either creates a subgoal, or selects a problem space, a state, or an operator to apply). The CPU time on an SGI Indigo R4000 is about 48 minutes.

The extended example demonstrates Instructo-Soar's variety of instructional capabilities, all arising from a single learning theory as it is applied to various instructions and learning situations. These capabilities operate together over the course of a complex instructional scenario to extend the agent's domain knowledge in a significant way, including learning each type of PSCM knowledge.

## 6.2 Limitations and opportunities for further research

This thesis is about *agents* that are *instructable* through *tutorial instruction*. Thus, the limitations of the work fall into three major categories. First, there are limitations inherent in the paradigm of tutorial instruction itself. Second, the agent being taught has limited capabilities. Third, there are limitations to the agent's instructability because of incomplete solutions to each of the problems faced by an instructable agent: the mapping problem, the interaction problem, and the transfer problem.

### 6.2.1 Limitations inherent in tutorial instruction

Tutorial instruction has been defined here as instruction that is both interactive and situated. This is a powerful knowledge source for a learning agent, because it provides knowledge interactively at the point of the agent's need that applies to particular situations the agent is facing or will face.

However, much instruction is either non-interactive or unsituated (or both). In non-interactive instruction, the content and flow of information to the student is controlled primarily by the in-

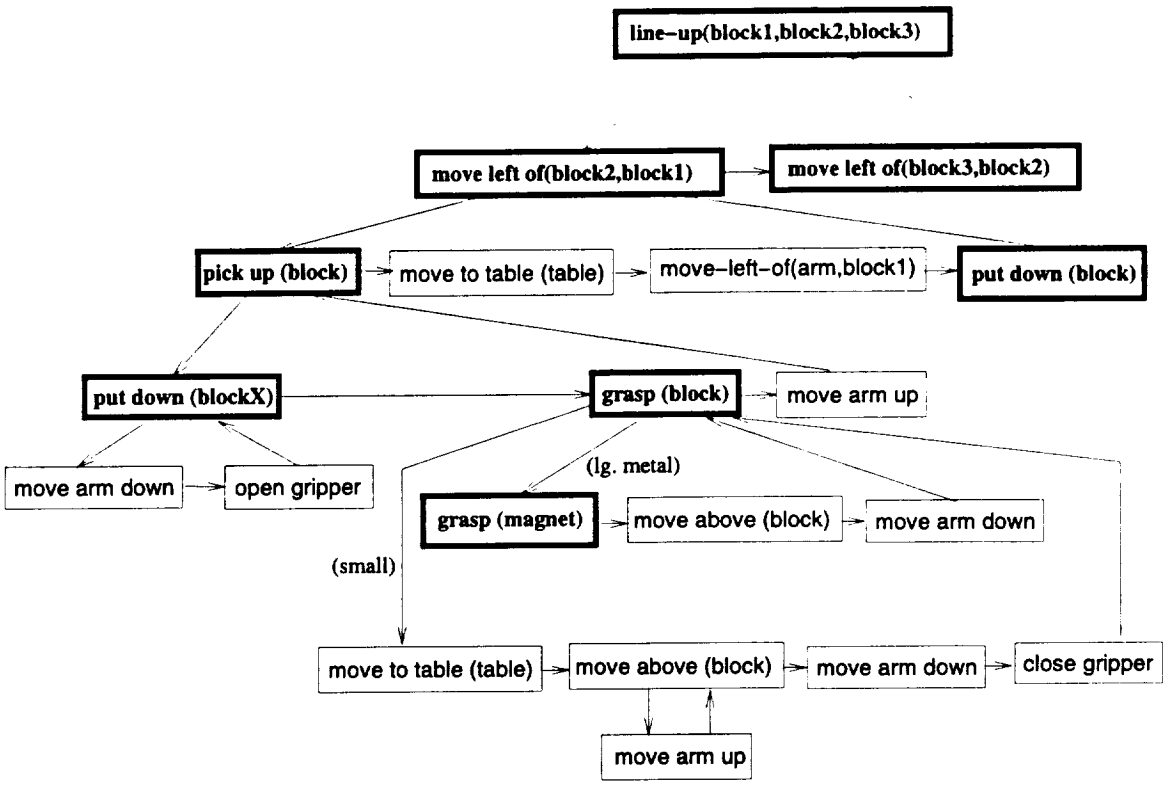



Figure 6.1: Part of the hierarchy of operators learned during the extended example of Appendix A.

formation source. Examples include classroom lectures, instruction manuals and textbooks. One issue in using this type of instruction is locating and extracting the information that is needed for a particular problem at hand. Some information conveyed non-interactively may be situated; e.g., in worked-out example problems [Chi *et al.*, 1989; VanLehn, 1987]. However, non-interactive instruction often includes unsituated information as well; e.g., general expository text.

Unsituated instruction conveys general or abstract knowledge that can be applied in a large number of different situations. Such general purpose knowledge is often described as "declarative" knowledge [Singley and Anderson, 1989]. For example, in physics class students are taught that $F = m \cdot a$; this general equation applies in specific ways to a great variety of situations. The advantage of unsituated instruction is precisely this ability to compactly communicate abstract knowledge that is broadly applicable [Sandberg and Wielinga, 1991]. Singley and Anderson [1989] point out that to use such abstract knowledge, students must learn *how* it applies to specific situations, thus transforming it from unsituated to situated knowledge. Their theory, described in Chapter 5, proposes that this transformation occurs as students solve specific problems, guided by unsituated knowledge gleaned from instruction.

A complete agent should be able to learn not only from tutorial instruction, but from non-interactive and unsituated forms of instruction as well. These other types of instruction have not been considered in this research.

### 6.2.2 Limitations of the agent

In addition to limitations inherent in tutorial instruction, an agent's own limitations constrain its instructability. The theory of learning from tutorial instruction developed here specifies a particular computational model in which to build agents (the problem space computational model); within this computational model, an agent with a particular set of capabilities has been implemented to demonstrate the theory. Thus, both the weaknesses of the computational model and the of the

specific implemented agent must be examined.

### Computational model

The problem space computational model plays a key role in the theory of tutorial learning. By defining the basic computational operations of an intelligent agent, it provides an enumeration of the types of knowledge that the agent uses, and thus should be able to learn from instruction. Its universal subgoaling nature supports flexible interaction and multiple levels of embedded instruction and learning.

The PSCM is well suited for situated instruction because of its inherently local control structure. However, this makes it difficult to learn tasks requiring global control regimes through instruction, because they must be translated into a series of local decisions that will each result in only local learning. If an instructable system will always be applied to problems requiring a *particular* global control method, it may be more effective (although less general) to build the control method into the system from the start. This is the approach of method-based knowledge acquisition tools [Birmingham and Klinker, 1993].

A second weakness of the PSCM is that it provides a theory of the *types* of knowledge used by an intelligent agent, but gives no indication of the possible *content* of that knowledge. A content theory of knowledge would be useful in specifying the complete space of possible instruction, to provide a finer grained analysis of instructability within the knowledge types analysis provided by the PSCM. Projects such as Cyc [Guha and Lenat, 1990] have the goal of developing detailed knowledge content theories.

Within the PSCM, an agent has been implemented with a particular set of basic capabilities, to develop and test the tutorial learning theory. Next, the limitations of this agent's capabilities are discussed.

### Implemented capabilities

The Instructo-Soar agent is limited in a number of ways that constrain its instructability:

- **Simple actions in a static, noise-free world.** The agent's actions each have a definitive starting point, and are defined by a single endpoint goal. This precludes actions with path constraints (e.g., "Go to the other room but don't walk on the carpeting!"). The world is noise-free in that actions always succeed (blocks never slip from the agent's hand), so it does not need to reason about failed actions.

- **Serial execution of actions toward a single top level goal.** Actions are executed serially, and the agent has only a single top level goal at a time, although there may be multiple embedded levels of operators selected to achieve it. This contrasts with dynamic domains such as flying an airplane, where multiple goals at multiple levels of granularity may be active at once [Pearson *et al.*, 1993]. In such a domain, an agent might be instructed to "Climb to 1000 feet while keeping the wings level, using a slow climb rate." While performing that action, the agent could be given yet another unrelated action to perform, such as "Raise your landing gear now."

- **Complete, perfect perception.** The agent has complete, noiseless perception of its environment. Thus, it never has to be told where to look, or asked to notice a feature that it overlooked (although non-perceivable features may be unknown). In human instruction, on the other hand, students must often be told where to attend or what features to notice, as shown by the protocol excerpt in Figure 6.2.

- **No history.** The agent does not keep track of past actions that have been performed (other than its episodic instruction memory) or states that have been seen. Thus, it cannot be asked to "do what you did before." Similarly, the agent cannot learn procedures that are defined by a particular sequence of actions to perform, rather than a set of state conditions to achieve.

| | Speaker | Instruction | Action |
|---|---|---|---|
| 137 | Instructor | This is actually the real runway we're crossing, but we can take off so fast... | taking off |
| 138 | | So, to maneuver the plane once we're flying... | in the air |
| 139 | | You see that our altitude is increasing, | points to altimeter |
| 140 | | our climb rate gauge has gone up, | points to climb rate gauge |
| 141 | | our speed has gone up | points to speedometer |
| 142 | | you know, the gauges change accordingly... | motions to gauges |
| 143 | | It's actually blue and brown... | points to false horizon |
| 144 | | This means we're going up at about a 10 degree angle. | points to line on false horizon |

Figure 6.2: An example of human instruction in which the instructor directs the student's perceptual attention.

For example, it cannot be taught how to dance, because dancing does not result in a net change to the external world.

- **No understanding of time or other resources.** The agent does not reason about the amount of time it takes to perform different tasks, or about other similar resources (e.g., its battery power). Thus, it cannot be taught to perform actions in a timely or resource-dependent way.

- **No search; e.g., no means-ends analysis.** Whenever the agent does not know what to do next, it asks for more instruction. It never tries to determine a solution through reasoning techniques like means-ends analysis. Human students, on the other hand, make extensive use of means-ends analysis and related methods [Newell and Simon, 1972]. Adding these capabilities to Instructo-Soar would produce a more realistic instructional dialogue, decreasing the agent's need for instruction.

- **Single agent.** There is only one agent in the domain. This precludes instruction teaching multiple agents to perform a task together (e.g., two robots carrying a couch), and instructions that require reasoning about other agents' potential actions (e.g., "Don't go down the alley, because if you do the enemy will block you in.")

Producing a definitive agent has not been the goal of this work. Rather, the capabilities of the agent have been developed to the point needed to provide a basic demonstration of each instructional learning and interaction capability.

In addition to the tutorial instruction paradigm and the agent's capabilities, Instructo-Soar's instructability is limited because its solutions to each of the key instructional learning problems – mapping, interaction, and transfer – are incomplete in various ways. The following sections discuss weaknesses and points for further research on each of these problems.

### 6.2.3 Mapping problem limitations

The mapping problem is the problem of mapping the language of instructions into the internal language used by the agent. Instructo-Soar employs a straightforward approach to mapping, leaving all of the problems of mapping difficult natural language constructions unaddressed. Some of the relevant problems, as described in Section 1.2.1, include reference resolution, incompleteness, and the use of domain knowledge in comprehension.

Instructo-Soar uses a simple matching process to resolve referents. Real instruction, of course, contains many difficult reference problems that require more complex methods. In addition, resolving a referent may require perceptual search (e.g., looking for "the block with a crack in it"), or even further instruction, as in:

```
> Grab the explosive block.
Which one is that?
   > The red one.
```

Instructo-Soar does not support this kind of interaction about referents.

Instructions that give incomplete information are common in human instructional dialogues. For instance, objects to be acted upon might be incompletely specified, as in "Line up three objects." Alternatively, an action itself may be incompletely specified, as in "Use the shovel" [Martin and Firby, 1991]. Instructo-Soar does not handle these kinds of incomplete instructions; it requires well specified actions with objects given in a particular order.

Finally, interpreting an instruction may require reasoning about the task itself. For example, DiEugenio and Webber [1992] point out that in interpreting "Cut the square in half to form two triangles," the correct cut can only be determined by reasoning about how it will produce the desired result. There has been some initial work on how NL-Soar might make use of domain knowledge during language comprehension [Lehman *et al.*, 1993], but Instructo-Soar does not do this kind of reasoning in interpreting instructions.

In addition to these general problems, Instructo-Soar does not support the full range of linguistic forms that might be used to communicate each of the types of PSCM knowledge that it can learn. For example, although the agent can learn about unknown effects of operators, it cannot be told them directly; e.g., "Doing X causes Y to happen." Similarly, the agent learns termination conditions for new operators after executing the operator, but cannot be told them directly; e.g., "A block is picked up when you are holding it and your arm is raised."

In fact, Instructo-Soar does not make full use of semantic information in creating a new operator. For example, when it first reads "Move the red block left of the yellow block," the agent creates a new operator, but does not make use of the semantic information communicated by "Move...to the left of." A more complete agent would try to glean any information it could from the semantics of an unfamiliar command.

### 6.2.4 Interaction problem limitations

The agent's shortcomings on the interaction problem center around the three requirements for solving the problem in a complete tutorable agent: (1) flexible initiation of instruction, (2) full flexibility of knowledge content, and (3) full situation flexibility. As Table 6.3 indicates, Instructo-Soar does not provide flexible instruction initiation, and gives a partial solution to knowledge content and situation flexibility.

A complete instructable agent should support instruction that is initiated by either the agent or the instructor. In Instructo-Soar, instruction is initiated only by the agent. This precludes giving instructions interrupting actions: "No! Don't push that button!"

A complete instructable agent should also allow flexible knowledge content in instructional interaction. That is, the agent should allow the instructor to communicate any piece of knowledge at any point that it is applicable within the current task and discourse context. Instructo-Soar fulfills this requirement for commands to be performed, but the interaction is more limited for other types of instructions.

Evaluating whether an instructable agent fulfills this requirement in general calls for a notion of *knowledge coherence*: a complete instructable agent must accept any instruction that communicates "coherent" knowledge at any point in the instructional dialogue. Defining knowledge coherence is not difficult for action commands; any command that may lead towards goal achievement is acceptable. Defining knowledge coherence for other types of knowledge is an area for further research. Work on discourse plans in task-oriented dialogues (e.g., [Litman and Allen, 1990; Litman and Allen, 1987; Allen and Perrault, 1980] may be useful in developing this notion.

Finally, a complete instructable agent should support full situation flexibility, allowing instructions that apply either to the current situation or to a specified hypothetical situation. Instructo-Soar meets this requirement by handling both implicitly and explicitly situated instructions, but hypothetical situations can only be referred to within a single instruction. Human tutors often refer to one hypothetical situation over the course of multiple instructions.

In Instructo-Soar, the situation an instruction applies to must be determined when that instruction is *received*. In human instruction, the goal of an action may be unknown when the action is performed [Redmond, 1992]. For instance, the instructor may lead a student through a number of actions and then say "We just built a tower." Currently, Instructo-Soar could not learn effectively from this kind of interaction; it must have the goal (building a tower) indicated in advance of the instructions for achieving it.

### 6.2.5 Transfer problem limitations

This work has focused most on solving the transfer problem – producing general learning from tutorial instruction. As Table 6.3 indicates, most of the transfer problem criteria for a complete tutorable agent have been achieved. The two criteria that have not been met are the ability to deal with incorrect knowledge, and the coexistence of instruction with learning from non-instruction sources. In addition, the methods that Instructo-Soar falls back on when it cannot explain instructions are not as powerful as they could be.

Instructo-Soar cannot recover from incorrect knowledge that leads to either invalid explanations or incorrect external performance. Such incorrect knowledge may be a part of the agent's initial domain theory, or may be learned through faulty instruction. The Einstellung effect, described in Chapter 5, is a non-fatal example of this; once the agent has learned an inefficient method for performing a task, it cannot go back and learn a more efficient method. More seriously, the agent cannot recover from learning an incorrect method.

In order to avoid learning anything incorrect, whenever Instructo-Soar attempts to induce new knowledge, it asks for the instructor's verification before adding the knowledge to its long-term memory. Human students do not ask for so much verification; apparently they are able to jump to incorrect conclusions, and then alter them later based on further information. Since Instructo-Soar remembers a single version of each piece of knowledge, inability to recover from incorrect knowledge rules out learning from multiple examples, which would require changing the single hypothesis (an alternative would be to explicitly represent multiple hypotheses; e.g., in a version space [Mitchell, 1982]). Similarly, it precludes instruction by general case and exceptions; for instance, "Never grasp red blocks," and then later, "It's ok to grasp the ones with safety signs on them."[1] Recovery from incorrect knowledge is discussed further below as one component of a specific proposal for future work.

The second transfer problem requirement that is not met by Instructo-Soar is instructional learning's coexistence with learning from other knowledge sources. Nothing in Instructo-Soar's theory or implementation precludes this coexistence, but it has not been produced since the focus of this work is instruction. Learning from other knowledge sources may even be invoked through instruction. For instance, the instructor may point to a set of objects and say "This is a tower," prompting learning from observation, or use an instruction containing an explicit metaphor to invoke analogical learning.

Finally, Instructo-Soar's solution to the transfer problem is limited by three aspects of its implementation. First, Instructo-Soar does attempt to form explanations of state inference instructions. This has not proven to be a problem for the simple state inferences learned thus far, but for more complex inferences, explanation may improve learning. For example, by explaining the instruction "If the switch is 'up', the control panel is powered," the agent could realize that the inference also depends on the panel being plugged in. Second, Instructo-Soar's initially rote learning of instructions is only recalled in situations *exactly* like the ones the original instructions were given in (until more general learning is completed); transfer might be improved by adding analogical recall. Third, the heuristics that Instructo-Soar uses in inducing missing knowledge to complete

[1] This example is due to Doug Pearson.

explanations (e.g., the **OP-to-G-path** heuristic) are very simple. More complex approaches, such as deeper causal theories or abductive reasoning, could lead to higher quality inductions.

## 6.3 The next generation instructable agent

Rather than attempt to address all of Instructo-Soar's limitations at once, this section proposes a "next generation" instructable agent that addresses a set of the most crucial limitations. As in the current work, the focus is on providing a broad range of flexible interaction and powerful instructional learning, while addressing the mapping problem sufficiently to support these other capabilities.

The proposed agent extends Instructo-Soar in four major ways:

1. **Use of means-ends analysis (agent capability).** Rather than always asking for instruction, the agent will attempt to solve problems it faces through means-ends analysis (and perhaps other simple search techniques), and only ask for help if this search fails. This will reduce the amount of instruction that the agent requires, making its interaction with its instructor more realistic. For example, when told to "Move the arm above the red block," the agent would discover via search that it must first move the arm up, instead of requiring further instruction for that step.

2. **Selective perception (agent capability).** Rather than complete perception of its world, the agent will have perception only of an attended region. The agent will perceive other areas by shifting its attention; attending closely to an object will uncover more of its perceptual details. This capability will allow the instruction techniques to be applied to more complex domains where complete perception is intractable and noiseless perception is unrealistic. Selective perception will permit instructions that direct perceptual attention to particular areas, objects, or features, as in "Look for the cracked block."

3. **Simple observational learning (transfer problem).** The agent will be able to learn by observing the effects of its actions. This is a simple way to learn operator effects (frame axioms) that can later be used in instruction explanations. In addition, the agent may learn by being directed to observe (e.g., "That is a tower").

4. **Learning by reaching conclusions and revising as needed (transfer problem).** Rather than always assuring that knowledge being learned is correct – either supported by prior knowledge (through explanation) or explicitly verified by the instructor – the agent will learn from "reasonable" inferences without verification. The agent will ask for verifications only in extreme cases. This does not require a major change to the current theory; it simply means that the agent will rely more on completing explanations through induction (option 2.1 from Table 2.4) and less on further instruction and verification (option 2.2). Since the inferred knowledge may be incorrect, however, the agent will require the ability to recover from incorrect knowledge. There has been some work on recovering from incorrect knowledge in Soar [Laird, 1988; Laird *et al.*, 1989], but the broad range of possible incorrectness in an instructable agent will require more research on basic recovery and theory revision techniques (as, e.g., [Ourston and Mooney, 1990; Ginsberg, 1988]).

The combination of these changes will produce an agent that has a much more realistic pattern of interaction with its instructor. The agent will require less instruction: it will solve tasks using MEA when it can, it will learn about its environment from observation as well as instruction, and it will make inductions without requiring continual verification. In addition, the agent itself will be more realistic, with limited perception that requires the instructor to direct its attention, just as would be required for a human student (or a real robot).

The new agent's learning will follow a different pattern than Instructo-Soar's. Instructo-Soar starts with an incomplete domain theory and extends it further and further through instruction. The new agent will start with a domain theory that may be both incomplete and incorrect. Rather

than simply building up, it will "jump around" in the space of theories, reaching conclusions that are beyond its current knowledge and instructional input, but then revising them in light of new information.

## 6.4 Conclusion

Instructo-Soar's most important result is its coverage of a crucial subset of the criteria for a complete tutorable agent. The key criteria that have been met include learning all types of knowledge used in task performance, providing fully flexible interaction for commands, and supporting both implicitly and explicitly situated instructions.

Empirical results demonstrate the agent's full range of instructional capabilities. While much work in machine learning aims for *depth* at a particular kind of learning, Instructo-Soar demonstrates *breadth* – of interaction with an instructor to learn a variety of types of knowledge – but all arising from one underlying technique. This kind of breadth is crucial in building an instructable agent, because of the great variety of instructions and the variety of knowledge that they can communicate.

Instructo-Soar has many weaknesses, arising from limitations of the tutorial instruction paradigm itself, limitations of the agent's capabilities, and limited solutions to the mapping, interaction, and transfer problems. The limitations indicate the kinds of steps that need to be taken to produce a more complete instructable agent. A specific proposal was made to produce the next generation of instructable agent by combining a particular set of the possible extensions.

# Chapter 7

# Conclusion

Current intelligent systems must be laboriously programmed for each new task they are to perform. This thesis has presented an alternative: teaching intelligent agents to perform tasks through tutorial instruction. Tutorial instruction is a powerful knowledge source for an intelligent agent. It is both interactive and situated, providing needed knowledge from an expert that applies directly to situations the agent is facing or will face. This type of instruction has been shown to be highly effective for human learners, but has received only limited attention in artificial intelligence and machine learning.

This thesis makes three major contributions to the study of tutorial instruction: (1) a task analysis of the tutorial learning task, (2) a general theory of learning from tutorial instructions, and (3) an implemented agent that embodies the theory, demonstrating a broad range of instructional learning and interaction capabilities.

First, an analysis of the task faced by a tutorable agent revealed the three problems that such an agent must solve: the mapping problem, the interaction problem, and the transfer problem. For each problem, a set of requirements for a complete solution was developed. Taken together, these requirements define the goal of the tutorial learning task, by indicating what is required of a "complete" tutorable agent. As such, they provide an operational evaluation criterion for work on tutorial learning, and serve to focus further research by indicating which parts of the problem have not been sufficiently addressed.

Second, the thesis presents a theory of learning from tutorial instruction. The theory has two parts: a computational model of an agent, and a learning method. The computational model, called the ***problem space computational model***, defines the learning task by specifying the types of knowledge that an agent uses in task performance, and, thus, that it must be able to learn from instruction. The learning method, called ***situated explanation***, specifies how the agent can learn general knowledge from instruction. For each instruction, the agent first determines what ***situation*** the instructed action or inference is intended to apply to. Then, it attempts to ***explain*** how the instruction leads to goal achievement in that situation, by internally projecting the instruction in the situation. This explanation process can involve prior domain knowledge, induction, and/or further instruction to complete an explanation.

Third, an implemented agent, Instructo-Soar, embodies the theory of tutorial learning and demonstrates its performance. The agent is built within Soar, a problem solving and learning architecture that follows the problem space computational model. By applying situated explanation to a wide range of instructions in different situations, Instructo-Soar demonstrates a breadth of interaction and learning capabilities, as is required by tutorial instruction. This breadth of capabilities is produced by a general learning method applied within a general agent, rather than by a grab-bag of specialized techniques. Instructo-Soar's instructional capabilities go beyond those of previous instructable systems (e.g., learning apprentice systems), meeting a number of the key requirements for tutorable agents. In particular, it is able to learn each of the types of knowledge it uses in task performance from instruction; it supports flexible interaction for action commands, allowing the instructor to skip steps and command new tasks at any point; and it can learn from

both from implicitly and explicitly situated instructions.

In addition, Instructo-Soar provides the potential basis for a cognitive model of learning from tutorial instruction. It makes a specific set of behavioral predictions that are broadly consistent with what is known about human behavior. An analysis of the model revealed how it was affected by its development within a unified theory of cognition, by factoring the impact of each of Soar's properties and entailments on behavioral properties of the model. The analysis illustrates a methodology for analyzing the effect of unified theories on microtheories developed within them.

# Appendix A

# An Example Instruction Scenario

This appendix describes Instructo-Soar's performance on a large instructional example. The example includes 100 instructions, and demonstrates nearly all of Instructo-Soar's learning and interaction capabilities (listed in Table 6.2). It includes examples of learning each of the types of PSCM knowledge, and examples of each type of interaction the agent can handle. Many of the examples in the body of the thesis are included in this example.

At the start of the example scenario, the Instructo-Soar agent has the initial knowledge described in Chapter 3. That is, it knows primitive operators for moving its arm up, down, and into relationship with other objects (e.g., above them); it can open and close its hand; and it can dock with tables. It does not know about the operation of the light or the electromagnet. In addition, before the example starts, the agent has been taught to put objects down (by moving down and opening its hand).[1] Throughout the scenario, the agent uses the "introspective" recall mode, attempting to recall and generalize new procedures as soon as they are taught (as described in Chapter 3).

## A.1 Example scenario

In what follows, the instructor's inputs are numbered, and shown in **typewriter** font; the agent's responses are shown in *italics*; and the agent's actions in the external environment are indicated by **boldface** print. Each of the agent's responses takes a simple, templated form; natural language generation has not been the focus of this work. The amount of indentation indicates the "depth" of an instruction in the interaction. That is, an instruction will be further indented if it teaches a procedure commanded by an earlier instruction; indentation is not a part of the actual agent's input. The world begins in the state shown in Figure A.1.

The instructor begins by teaching the agent some of the properties of magnets:

**1> Red magnets are off.**
*Gotcha.*
**2> White magnets are powered.**
*Gotcha.*

From each of these, the agent learns a state inference rule, as described in Section 4.1.1. In the current state, the magnet is red, so the appropriate rule applies to conclude that the magnet has **status off**.

Next, the instructor teaches the agent a new procedure.

**3> Pick up the red block.**
*That's a new one for me. How do I do that?*

[1] This could have been included within the example (e.g., when the agent is first asked to put down a block). However, "put down" is used to demonstrate extending another procedure (picking up something when holding something else), and this is clearer if "put down" is already known.

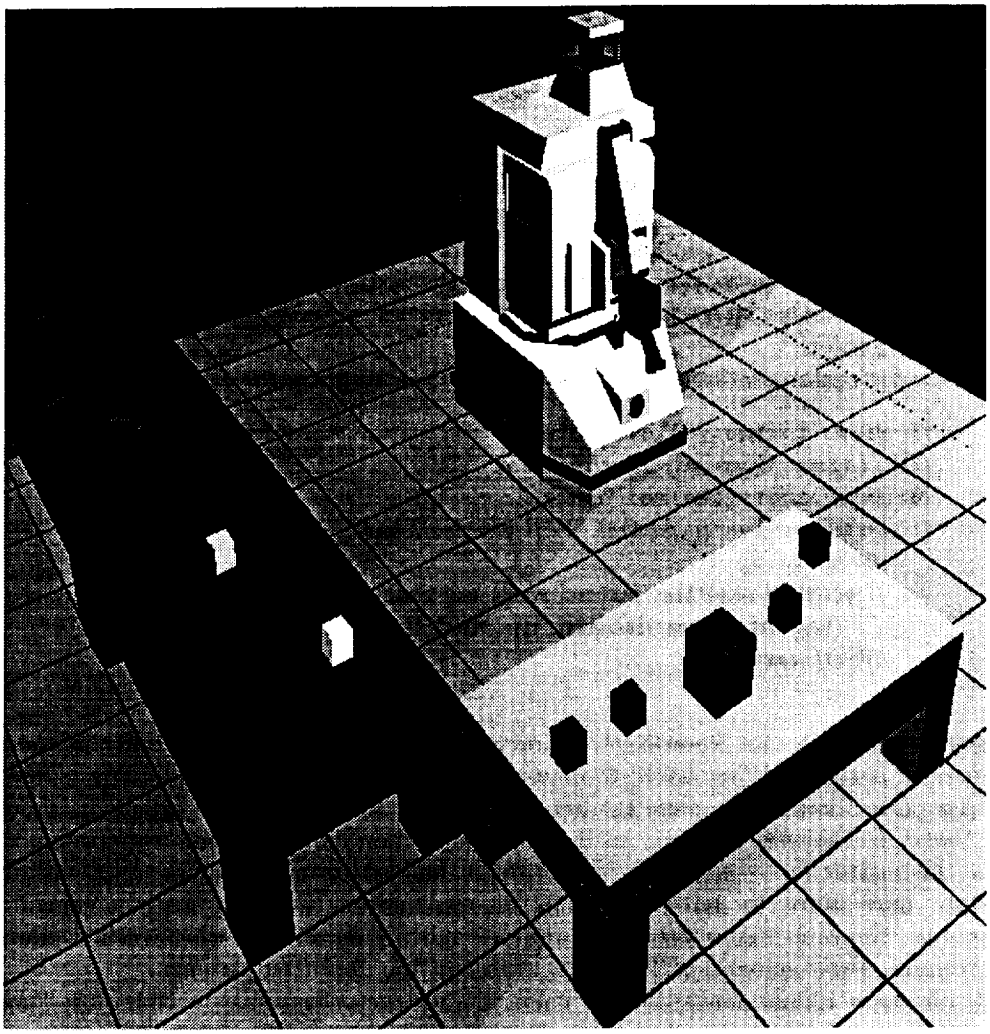


Figure A.1: The initial world state for the example instruction scenario.

The agent realizes this operator is unknown, and generates a new operator template and natural language mapping for the operator, as described in Section 3.4. Then, it asks for more instructions.

**4> Grasp the red block.**
*That's a new one for me. How do I do that?*

Here, in instructing one new procedure, the instructor has asked the agent to perform another new procedure. The agent again creates a new operator template, and asks for further instruction on how to grasp a block.

**5> If the red block is metal then grasp the magnet.**
*So I'm guessing the conditions for doing "grasp the magnet"*
*when your goal is "grasp the block" are:*
*the block is metal*
*Is that right?*
**6> Right.**

Instruction 5 indicates a contingency action for grasping metal blocks. The red block is not metal, but the instructor wants the agent to know what it should do if asked to grasp a metal block. The agent's response, as described in Section 4.1.3, is to try to induce the conditions under which the contingency action should be taken, and then ask the instructor to verify its induction. The rule that is learned from this interaction, shown in Figure 4.6, proposes grasping the magnet in order to grasp a metal block.

Now, the instructor continues to teach the agent how to grasp the red block.

**7> Move to the yellow table.**
*Ok.* **(agent does so)** *What next?*
**8> Move above the red block.**
*I don't know how to do that. Will you teach me?*
**9> Move up.**
*I get it! Doing that lets me reach my goal.*
**(agent moves its arm up, and then above the red block.)**
*What next?*

The agent begins with the knowledge of how to move above objects, but only in the situation where its arm is raised. When asked to move above the red block, the agent's arm is lowered, and it does not know how to complete the action. Thus, it asks for help to extend its knowledge of "move above" to this new situation, as described in Section 3.5. When told to move up, the agent explains this instruction by internally projecting it, and realizes after moving up, it will be able to successfully move above the red block. From this explanation, the agent learns a general operator proposal rule, shown in Figure 3.15, that proposes moving the arm up whenever it is down and its goal is to move above something. Then, it carries out the instructed actions.

With the agent's hand above the red block, the instructor continues teaching the "grasp" procedure.

**10> Move the arm down.**
*Ok.* **(agent does so)** *What next?*
**11> Close the hand.**
*Ok.* **(agent does so)** *What next?*
**12> The operator is finished.**

As described in Section 3.4.1, when the instructor indicates that the "grasp" operator is finished, the agent attempts to induce termination conditions for the operator. This is done by comparing the current state of the world, shown in Figure A.2, to the initial state (Figure A.1). The induction is presented to the instructor for alteration and verification:

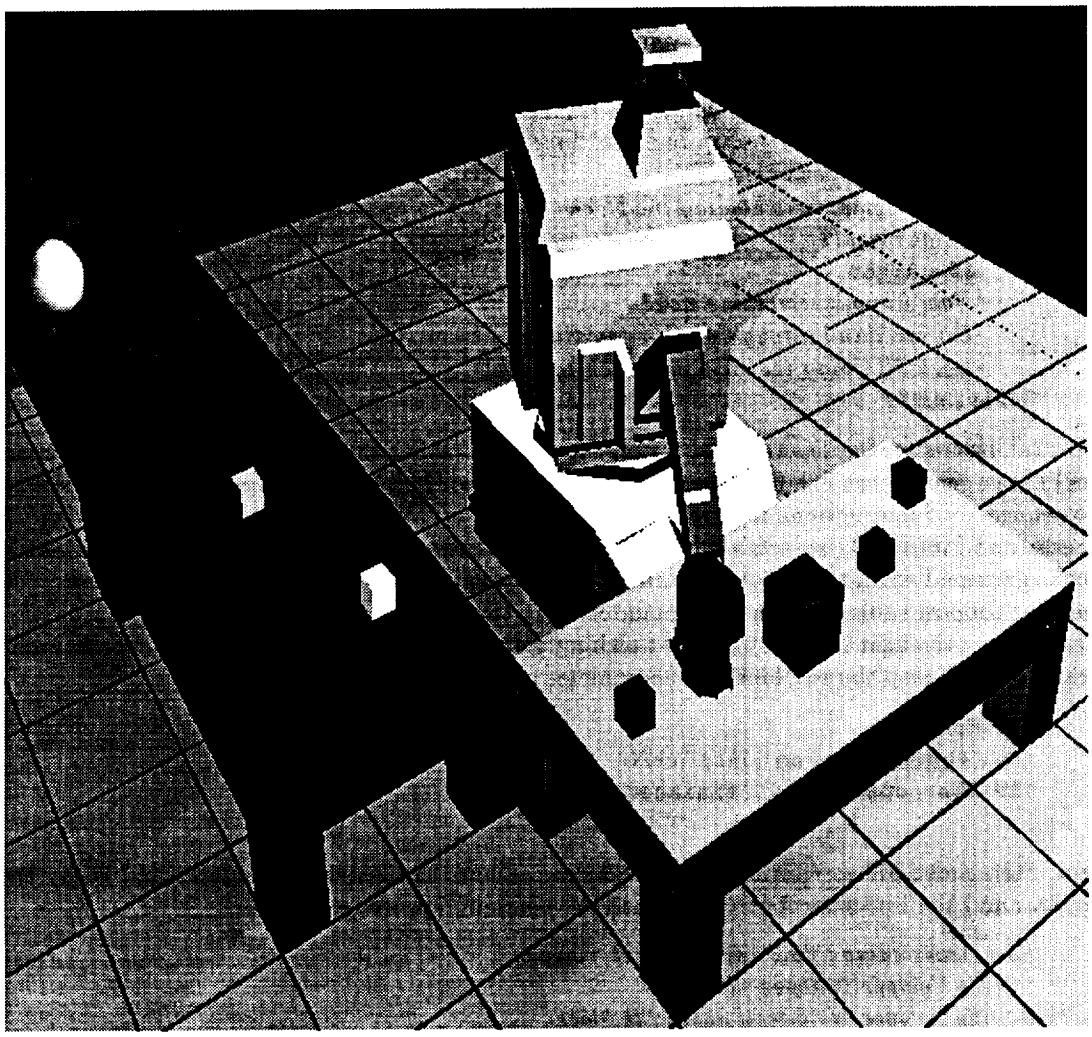


Figure A.2: The state of the world after grasping the red block.

*So I guess doing "grasp the block" means:*
*the gripper is above the block*
*the gripper is directly above the block*
*the gripper is closed*
*the gripper is holding the block*
*the robot is docked at the table*
*Is that right?*
**13> The robot need-not-be docked-at the yellow table.**
*So I guess doing "grasp the block" means:*
...
*Is that right?*
**14> The gripper need-not-be above the red block.**
...
**15> The gripper need-not-be directly-above the red block.**
*So I guess doing "grasp the block" means:*
*the gripper is closed*
*the gripper is holding the block*
*Is that right?*
**16> Right.**
*Now I know something new!*
*Let me think through what we just did...*
*I get it! Doing that lets me reach my goal!*
*What next?*

After the instructor verifies the goal concept for "grasp" (that the gripper is closed and holding the block), the agent learns chunks that encode that knowledge, and then tries to recall and explain the sequence of instructions for grasping. Starting internally from the initial state (Figure A.1), it recalls and internally projects each command for performing "grasp the red block." Finally, the block is grasped within the projection, and from this explanation, the agent is able to learn general operator proposal rules for each command leading to grasping.

However, the agent has not finished picking up the red block. Now that the first command for picking up ("Grasp the red block") is completed, it asks what to do next.

**17> Move up.**
*Ok.* **(agent does so)** *What next?*
**18> The operator is finished.**

Again, the agent induces termination conditions, allows the instructor to alter and verify the induction, and then introspects to generalize the "pick up" procedure:

*So I guess doing "pick up the block" means:*
*the gripper is above the block*
*the gripper is directly above the block*
*the gripper is closed*
*the gripper is holding the block*
*the gripper is raised*
*the block is raised*
*the robot is docked at the table*
*Is that right?*
**19> The robot need-not-be docked-at the yellow table.**
...
**20> The gripper need-not-be above the red block.**
...
**21> The gripper need-not-be directly-above the red block.**
*So I guess doing "pick up the block" means:*

*the gripper is closed*
*the gripper is holding the block*
*the gripper is raised*
*the block is raised*
*Is that right?*
22> Right.
*Now I know something new!*
*Let me think through what we just did...*
*I get it! Doing that lets me reach my goal!*
*What should I do now?*

The agent is now holding the red block, and has learned a general procedure for picking up blocks when its hand is empty to begin with. Next, this procedure is extended to apply to the situation in which the agent is holding another object already.

23> Pick up the yellow block.
*I don't know how to do that. Will you teach me?*
24> Put the red block down.
*I get it! Doing that lets me reach my goal.*
**(agent puts the red block down, goes to the blue table, and picks up the yellow block.)**
*What next?*

By internally projecting the effects of putting down the red block, the agent realizes that this action enables it to pick up the yellow block, and learns operator proposal knowledge for putting an object down in order to pick up another one.

Next, the agent is taught another new procedure: pushing a button.

25> Put down the yellow block.
*Ok.* **(agent does so)** *What next?*
26> Push the green button.
*That's a new one for me. How do I do that?*
27> Move to the grey table.
*Ok.* **(agent does so)** *What next?*
28> Move above the green button.
*Ok.* **(agent moves up, and then above the block)** *What next?*

Notice that to move above the green button the agent applies what it learned earlier when moving above the red block – it moves up first.

29> Move down.
*Ok.* **(agent does so)** *What next?*
30> The operator is finished.
*So I guess doing "push the button" means:*
*the gripper is above the button*
*the gripper is directly-above the button*
*the robot is docked-at the table*
*Is that right?*
31> Right.
*Now I know something new!*
*Let me think through what we just did...*
*I get it! Doing that lets me reach my goal!*
*What should I do now?*
32> Go home.
*Ok.* **(agent does so)** *What next?*

Next, the instructor teaches the agent how to turn on the light and magnet, but rather than actually having the agent turn them on, hypothetical goals are used.

**33> To turn on the magnet, grasp the magnet.**

As described in Section 4.2, to learn from this type of instruction, the agent creates a hypothetical situation with the goal of turning on the magnet, and then projects the commanded action (grasping the magnet) to see if the goal is achieved. In this case, the agent has no knowledge of the effect of grasping the magnet on its status, and so cannot explain the instruction. Thus, using the heuristics described in Section 4.2.2, the agent guesses that grasping the magnet must cause it to turn on. The instructor is asked to verify this induction:

*Does this mean that an effect of "grasp the magnet" is that:*
*the magnet is now on*
*Is that right?*
**34> Yes.**
*I think that doing grasp the magnet causes:*
*the magnet to be on*
*under the following conditions:*
*the magnet is not on*
*the magnet is off*
*Are those the correct conditions?*
**35> Right.**
*Thanks for that advice, teach.*

Once the effect of "grasp the magnet" is learned, the agent is able to complete its explanation of instruction 33, and learns to propose grasping the magnet if its goal is to turn the magnet on. A similar exchange occurs for turning on the light:

**36> To turn on the light, push the red button.**
*Does this mean that an effect of "push the button" is that:*
*the light is now on*
*Is that right?*
**37> Yes.**
*I think that doing push the button causes:*
*the light to be on*
*under the following conditions:*
*the light is not on*
*the light is on the table*
*the button is on the table*
*Are those the correct conditions?*
**38> The red button must-be red.**
*I think that doing push the button causes:*
*the light to be on*
*under the following conditions:*
*the light is not on*
*the light is on the table*
*the button is on the table*
*the button is red*
*Are those the correct conditions?*
**39> Right.**
*Thanks for that advice, teach. What next?*
**40> Turn on the light.**
*Ok.* **(agent pushes the red button, and the light comes on)**
*What next?*

In this case, the instructor must indicate that the button to be pushed must be the red button. The other button (the green one) has a different effect; it does not cause the light to come on.

The next instruction teaches the agent to always keep the light on, using hypothetical state (a conditional). When the agent cannot explain why this is the proper course of action, it asks the instructor why:

```
41> If the light is off then turn on the light.
    Can you tell me why?
    42> Trust me.
Thanks for that advice, teach. What next?
```

Since the instructor declines to give an explanation, the agent heuristically learns to propose turning on the light whenever it is off, as described in Section 4.1.2.

Next, the agent is taught to control the light's brightness. It is taught new procedures for brightening and dimming the light by pushing the green button. However, since it does not know about the green button's effect on the light, it cannot form a complete explanation of these procedures, and thus cannot learn them in general form at this point.

```
43> Brighten the light.
That's a new one for me. How do I do that?
    44> Push the green button.
    Ok. (agent does so) What next?
    45> The operator is finished.
    So I guess doing "brighten the light" means:
        the gripper is above the button
        the gripper is directly-above the button
        the light is bright
    Is that right?
    46> The gripper need-not-be directly-above the green button.
        ...
    47> The gripper need-not-be above the green button.
    So I guess doing "brighten the light" means:
        the light is bright
    Is that right?
    48> Right.
    Let me think through what we just did...
    My introspection failed.
What next?
49> Move up.
Ok. (agent does so) What next?
50> Dim the light.
That's a new one for me. How do I do that?
    51> Push the green button.
    Ok. (agent does so) What next?
    52> The operator is finished.
    So I guess doing "dim the light" means:
        the gripper is directly-above the button
        the gripper is lowered
        the light is bright
    Is that right?
    53> The gripper need-not-be directly-above the green button.
        ...
    54> The gripper need-not-be lowered.
    So I guess doing "dim the light" means:
        the light is dim
```

*Is that right?*
**55> Right.**
*Let me think through what we just did...*
*My introspection failed.*
**56> Move up.**
*Ok.* **(agent does so)** *What next?*

Now that the agent knows how to brighten and dim the light, the instructor teaches it that the light should be kept dim. To explain why, the instructor points out that dim lights require less electricity. This example is described in Section 4.1.2.

**57> If the light is bright then dim the light.**
*Can you tell me why?*
**58> Dimmed lights use little electricity.**
*Gotcha.*
**59> Bright lights use much electricity.**
*Gotcha.*
*Thanks for that advice, teach. What next?*

By recognizing the difference in use of resources between bright and dim lights, the agent understands why the light should be kept dim. To test the agent's learning, at this point the light is made bright. The agent immediately selects the operator it has learned to "dim the light":

**(light becomes bright)**
**(agent selects operator to dim light)**

However, the agent did not learn a general implementation for that operator. Thus, after "dim the light" is selected, the agent recalls its past instruction, that it is supposed to "push the green button." Since its explanation of this instruction has failed in the past, the agent tries to induce general conditions for pushing the green button in this situation, using the heuristics described in Section 3.6.

*So I'm guessing the conditions for doing "push the button"*
*when your goal is "dim the light" are:*
*the gripper is not directly above the button*
*Is that right?*
**60> Right.**
**(the agent pushes the green button, and the light is dimmed.)**

Next, the instructor will teach the agent about the effect of magnetism, in grasping metal blocks using the magnet. First, the agent is taught two state inferences:

**61> Grey blocks are metal.**
*Gotcha.*
**62> If the magnet is directly-above a block and**
**the magnet is powered and the block is metal**
**then the magnet is stuck-to the block.**
*Gotcha.*

The rule learned from instruction 62 is shown in Figure 4.2.

In order to teach the agent about grasping blocks with the magnet, it must be taught how the magnet can cause a block to be "held". That is, the agent must learn that an effect of moving down when holding the magnet above a metal block is that the block becomes "held."

To teach this, the instructor must move the agent into the proper situation, shown in Figure A.3, and then use a hypothetical goal instruction so that the agent can induce a new operator effect for moving the arm down. The awkwardness of teaching this example reveals the limitations of Instructo-Soar's interaction capabilities for non-action commands. The example is described further in Section 4.2.3.

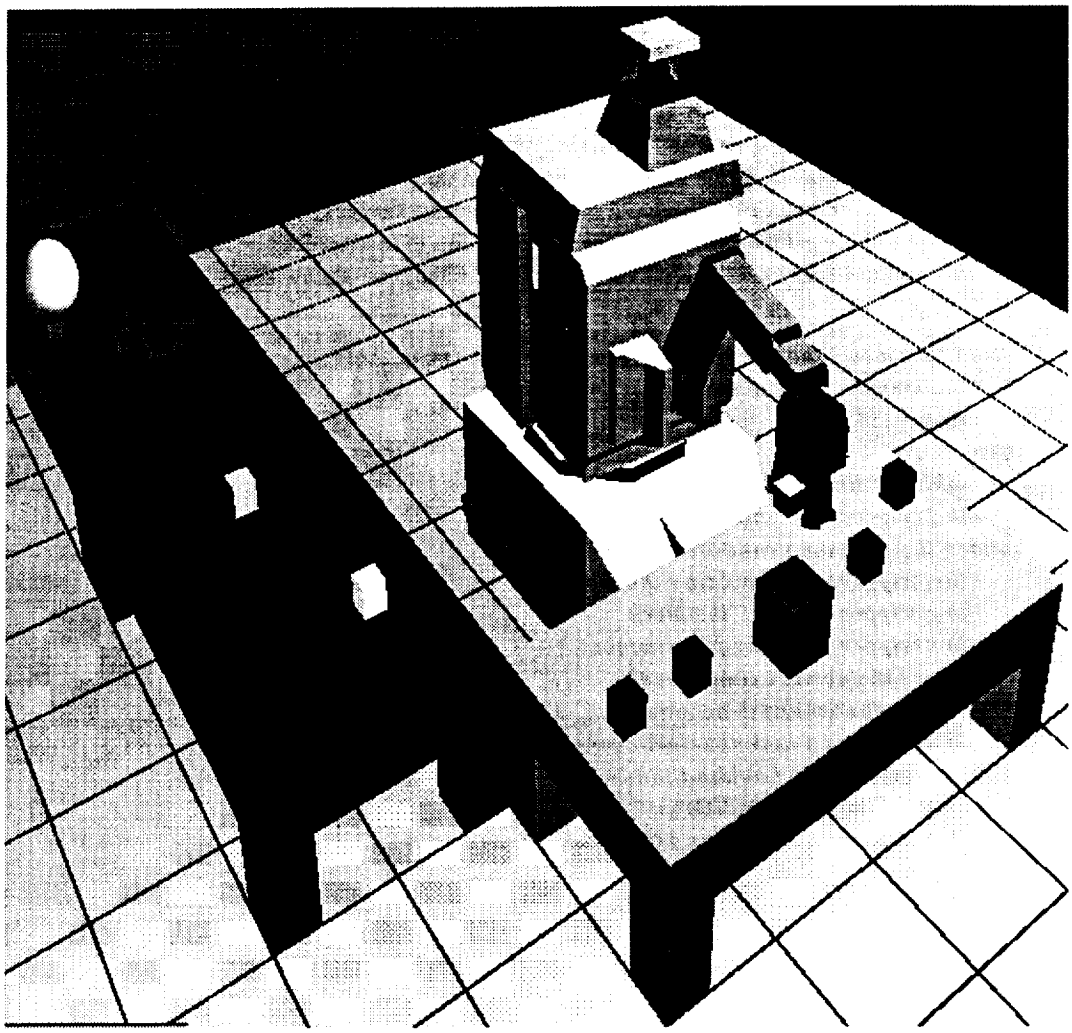


Figure A.3: The initial state the agent is placed in for learning about grasping by using a magnet. The agent is holding the magnet, and the magnet is on.

63> Grasp the magnet.
*Ok.* (agent does so) *What next?*
64> Move above the grey block.
*I don't know how to do that. Will you teach me?*
65> Move to the yellow table.
*I get it! Doing that lets me reach my goal.*
**(agent moves to the yellow table, and then the hand is moved up and then above the grey block.)**
*What next?*
66> To grasp the grey block, move down.

Along the way to getting the agent into the situation of Figure A.3, the instructor extends the agent's knowledge of moving above things to include the case where the agent is not at the right table when asked to move above something (instructions 64 and 65).

When told "To grasp the grey block, move down," the agent creates a hypothetical situation with a state like the current state, and a goal of grasping the grey block. It attempts to explain why moving down achieves this goal, but the explanation is incomplete because the agent does not know that the magnet causes the block to be "held." Thus, the agent infers a new effect of moving down:

*Does this mean that an effect of lowering the arm is that:*
*the gripper is now holding the block*
*Is that right?*
67> Yes.
*I think that doing lower the arm causes:*
*the gripper to be holding the block*
*under the following conditions:*
*the gripper is not holding the block*
*the gripper is above the block*
*the gripper is holding the magnet*
*Are those the correct conditions?*
68> The magnet must-be powered.
*I think that doing lower the arm causes:*
*the gripper to be holding the block*
*under the following conditions:*
*the gripper is not holding the block*
*the gripper is above the block*
*the gripper is holding the magnet*
*the magnet is on*
*Are those the correct conditions?*
69> Right.

As described in Section 4.2.3, this interaction allows the agent to learn a rule that simulates the new effect of moving down within internal projections. However, the agent must still learn to conclude that it is holding the metal block in actual execution. When asked to actually grasp the grey block, the agent expects to succeed after moving down, but does not, because it does not realize that it is "holding" the block with the magnet. As a result, it attempts to infer a new state inference that will complete grasping of the block.

70> Grasp the grey block.
**(agent moves the arm down, and the block gets stuck to the magnet.)**
*So, can I infer that:*
*the gripper is holding the block*
*Is that right?*

71> **Yes.**
*Why is the gripper holding the block now?*
72> **Because the gripper is holding the magnet and the magnet is stuck-to the grey block.**
*Ok. What next?*
73> **Go home.**
*Ok.* **(agent does so)** *What next?*

From this exchange, the agent learns to recognize that a block is held if it is stuck to the magnet, and the magnet is held. This rule is shown in Figure 4.18.

Before the instructor can give further instructions, the light shuts off unexpectedly. The agent responds by pushing the red button to turn the light back on.

**(light goes off)**
**(agent moves to table, pushes red button, and the light comes on.)**
*What next?*

At this point, the agent has learned how to pick up metal blocks using the magnet, and how to pick up small blocks using its hand. When asked to pick up a small metal block (the blue block), it has a conflict between these options, and asks for instruction to resolve the conflict. This capability is described in Section 4.3.

74> **The blue block is metal.**
*Gotcha.*
75> **Pick up the blue block.**
*These possibilities come to mind:*
*move to the yellow table*
*grasp the magnet*
*What should I do?*
76> **Move to the yellow table.**
*Ok.* **(agent does so)**
*These possibilities come to mind:*
*move above the block*
*grasp the magnet*
*What should I do?*
77> **Move above the blue block.**
*Ok.* **(agent does so, and then moves down.)**
*These possibilities come to mind:*
*close the hand*
*grasp the magnet*
*What should I do?*
78> **Close the hand.**
*Ok.* **(agent does so)**
*What next?*

From this interaction the agent learns a series of operator comparison rules that prefer the correct sequence of operators for grasping a small metal block by using the hand.

The next exchange illustrates learning about prohibited actions, as described in Section 4.1.4. The instructor teaches the agent not to grasp green blocks. When the agent cannot understand why, the instructor tells it that green blocks are explosive, allowing the agent to learn not to grasp anything that is explosive.

79> **Never grasp green blocks.**
*Why?*
80> **Green blocks are explosive.**
*Thanks for that advice, teach. What next?*

**81> Grasp the green block.**
*I cannot grasp the block because I know it won't work.*

The agent has also learned to avoid the *state* in which it is grasping an explosive object. When led to that state without being told to "grasp" explicitly, the agent grasps the explosive object momentarily, realizes its error, and recovers from it. It also learns to avoid the action that led to the bad state – here, to avoid closing its hand around an explosive object.

**82> Open the hand.**
*Ok.* **(agent does so)** *What next?*
**83> Move above the green block.**
*Ok.* **(agent does so)** *What next?*
**84> Move down.**
*Ok.* **(agent does so)** *What next?*
**85> Close the hand.**
*Ok.* **(agent does so)**
*This is a bad state to be in!*
**(agent opens its hand)**
*What next?*

The generality of the agent's learning can be tested by asking it to pick up another explosive object besides the green block.

**86> Blue blocks are explosive.**
*Gotcha.*
**87> Pick up the blue block.**
*No way!*

"Pick up the blue block" has grasping the blue block as a sub-operation. As soon as grasping the block is proposed, the agent realizes it will not be able to complete the "pick up" task, and abandons it.

Finally, to round out the hierarchy of new operators the agent has learned, the instructor teaches it to line up rows of three blocks; in the course of doing that, the agent is taught to place one block left of another.

**88> Line-up the white block, the red block and the yellow block.**
*That's a new one for me. How do I do that?*
**89> Move the red block left of the white block.**
*That's a new one for me. How do I do that?*
**90> Pick up the red block.**
*Ok.* **(agent does so)** *What next?*
**91> Move to the blue table.**
*Ok.* **(agent does so)** *What next?*
**92> Move the arm left of the white block.**
*Ok.* **(agent does so)** *What next?*
**93> Put down the red block.**
*Ok.* **(agent does so)** *What next?*
**94> The operator is finished.**
*So I guess doing move blocks left of one another means:*
[...many features...]
*Is that right?*
**95> Right.**
*Let me think through what we just did...*
*I get it! Doing that lets me reach my goal.*
*What next?*

The agent has just learned how to move one block left of another when the blocks start out on different tables. Therefore, the operator proposal rule that it learned for picking up the first block to move it left of the second one depends on the first one being on another table. When asked to move the yellow block left of the red one, the blocks are on the same table, so the agent must be told again to pick up the first block. This extends the "move left of" procedure to this new situation.

```
96> Move the yellow block left of the red block.
```
*I don't know how to do that. Will you teach me?*
```
   97> Pick up the yellow block.
```
*I get it! Doing that lets me reach my goal.*
**(agent picks up the yellow block and places it left of the red one.)**
*What next?*

This illustrates one of the problems with such a situated learning process: the agent only learns about the current situation, and must learn about others as they arise during later executions. The alternative would be a process that considered many possible situations at learning time.

```
98> The operator is finished.
```
*So I guess doing "line up blocks" means:*
[...many features...]
*Is that right?*
```
99> Right.
```
*Let me think through what we just did...*
*I get it! Doing that lets me reach my goal.*
```
100> Line-up the blue block, the white block and the purple block.
```
*Ok.* **(agent does so)** *What next?*

This final command takes seventeen primitive actions to perform. The final state is shown in Figure A.4.

## A.2 Summary of Results

Instructo-Soar begins with 2,579 productions. Through the course of this example (including learning the procedure "put down" before the scenario begins), the agent learns 4,730 chunks, for a final total of 7,309 productions. The example run takes 13,820 Soar decision cycles, including the application of 7,730 operators; the CPU time on an SGI Indigo R4000 is about 48 minutes.

Table A.1 indicates the number of examples of each of Instructo-Soar's instructional capabilities that are included in the scenario. Eighteen of its nineteen instructional capabilities are included; learning indifference between operators is left out.

The instructions given in this example can be classified in various ways. There were five different linguistic forms of instructions, as shown in Table A.2. The instructions can also be classified in terms of their interactional properties. These include whether the instruction is explicitly situated (in various ways), implicitly situated, or used to alter/verify of a piece of inferred knowledge. This classification is given in Table A.3.

The example involved learning each of the types of PSCM knowledge. Table A.4 indicates the amount learned of each type. Eight new procedures were learned, built up hierarchically as shown in Figure 6.1.

Table A.5 gives a finer grained view of the learning that took place by classifying the individual chunks that were learned. The table shows the number of chunks for various types of domain knowledge, for episodic memory for natural language comprehension, and for "bookkeeping" of various types. Note that the number of chunks does not directly correspond to the amount of each type of PSCM knowledge learned, because sometimes multiple chunks encode what can be viewed as a single piece of PSCM knowledge. The largest category of chunks by far were those associated with episodic memory. This is because of the factored representation of instructional episodes in

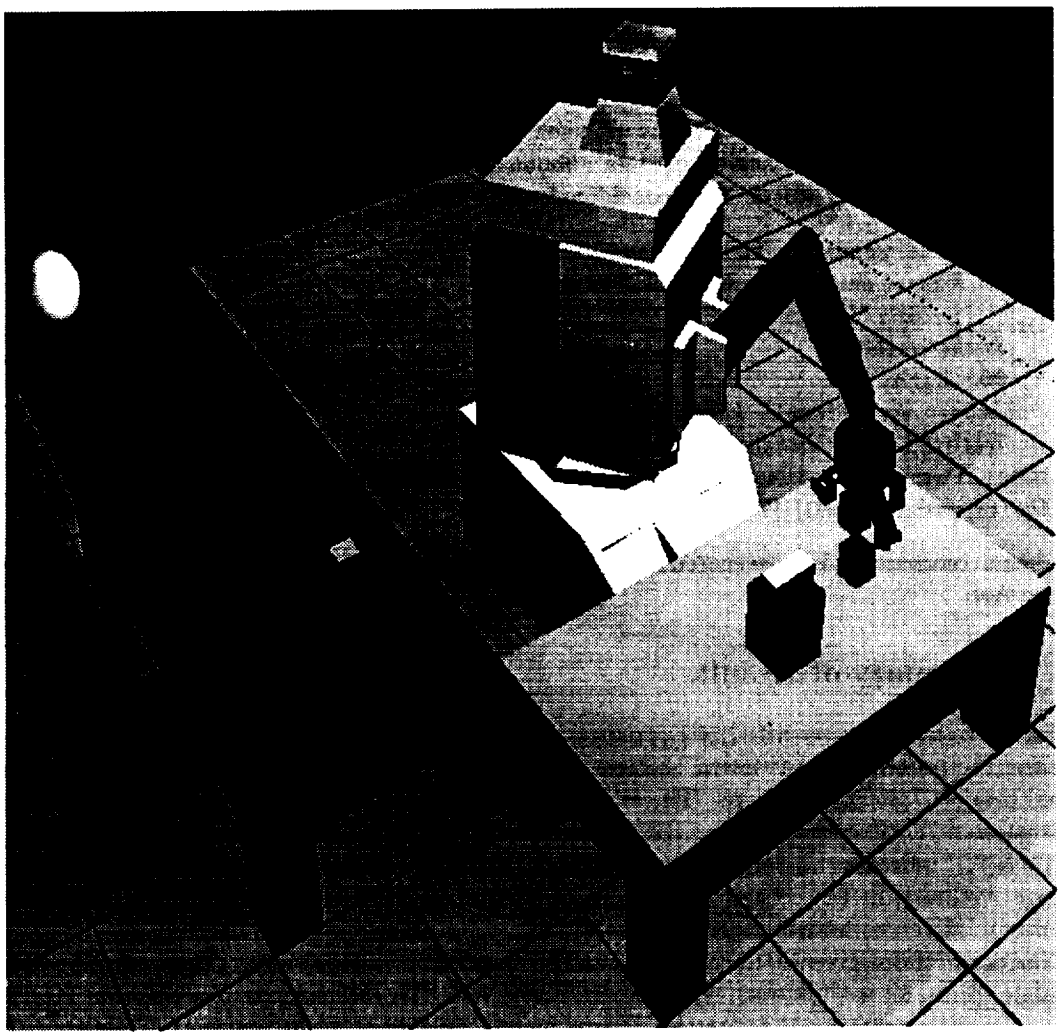


Figure A.4: The final state of the agent in the example scenario.

| | Instructional capability | Number |
|---|---|---|
| 1. | Learning completely new procedures | 8 |
| 2. | Taking instruction to alter/verify newly inferred knowledge (e.g. new operator's termination conditions) | 15 |
| 3. | Extending a known procedure to apply in a new situation | 4 |
| 4. | Hierarchical instruction: instructions for a new procedure embedded in other instruction | 2 |
| 5. | Prohibited actions | 1 |
| 6. | Explanations of prohibited actions | 1 |
| 7. | Recovery from indirect achievement of a prohibited state | 1 |
| 8. | Inferences from simple statements – specific | 1 |
| 9. | Inferences from simple statements – generic | 7 |
| 10. | Inferences from conditionals | 1 |
| 11. | Operators to perform for hypothetical goals | 3 |
| 12. | Contingency operators to perform in related situations | 1 |
| 13. | Operators to perform in hypothetical states | 3 |
| 14. | Explanations of operators for hypothetical states | 1 |
| 15. | Learning perceivable operator effects | 2 |
| 16. | Learning non-perceivable operator effects, and associated inferences to recognize the effects | 1 |
| 17. | Learning procedures in spite of an incomplete domain theory | 1 |
| 18. | Dealing with operator conflicts: learning which operator to prefer | 3 |
| 19. | Dealing with operator conflicts: learning choices are indifferent | 0 |

Table A.1: Instructional capabilities demonstrated in the extended example.

| Form | Example | Number |
|---|---|---|
| Imperative commands | "Move to the grey table." | 49 |
| Simple statements | "White magnets are powered." | 28 |
| Conditionals | "If the light is on then turn off the light." | 4 |
| Purpose clauses | "To turn on the light, push the red button." | 3 |
| Single words | "Right." | 16 |

Table A.2: Count of linguistic forms of instructions appearing in the extended example.

| Category | Number |
|---|---|
| Implicitly situated instructions | 54 |
| Explicitly situated instructions: | 16 |
| Hypothetical states | 13 |
| simple statements | 8 |
| conditional inferences | 1 |
| conditional actions | 3 |
| prohibited actions | 1 |
| Hypothetical goals | 3 |
| Verificational instructions | 30 |

Table A.3: A classification of the interactive properties of instructions appearing in the extended example.

| Type | Number |
|---|---|
| State inferences | 10 |
| Operator proposals | 36 |
| Operator rejections | 2 |
| Operator comparisons | 3 |
| Operator effects | 3 |
| Operator termination conditions | 8 |
| *Total* | 57 |

Table A.4: The PSCM types of knowledge learned during the extended example.

| | Chunk type | Number learned |
|---|---|---|
| Domain Knowledge | New operator templates (3/new operator) | 24 |
| | New NL mappings (2/new operator) | 16 |
| | Operator termination conditions (positive and negative tests) | 37 |
| | Operator proposals | 36 |
| | State inferences | 25 |
| | Operator effects | 4 |
| | Operator comparisons | 3 |
| | Operator rejections | 2 |
| Other Knowledge | Episodic memory | 2936 |
| | Other natural language | 1450 |
| | Associated with verification | 39 |
| | Assorted bookkeeping and side effects | 158 |
| | *Total:* | 4730 |

Table A.5: Types of chunks learned during the extended example.

long-term memory, in which each feature of each natural language referent is stored in a separate chunk, as described in Chapter 5.

# Appendix B

# A Mathematical Analysis of Instruction Sequences

One interaction requirement of tutorial instruction is the ability to accept any command that communicates knowledge relevant to the task at hand, at any point. Instructo-Soar provides this kind of flexibility for commands that indicate actions or tasks to perform. In teaching a procedure, instructors do not always specify the primitive operators that make up the procedure in their execution order. Rather, they form larger subtasks, indicate future operators before they can apply, and make assumptions about the agent's ability to reach future states without needing instructions. Supporting this kind of instruction means allowing the instructor to skip steps or command subtasks whether the missing steps or the commanded subtasks are known to the agent or not.

This appendix addresses the question of how much flexibility this kind of interaction allows the instructor. How variable can the instructions for a procedure be? That is, given a sequence of physical operations to be carried out in performing a particular procedure, how many sequences of instructions will produce that sequence of physical operations?

As will be shown, there are a large number of instruction sequences that can lead the agent down the same path to a given goal. That is, the mapping from instruction sequences to execution sequences is many to one. The number of instruction sequences possible for a given execution sequence can be derived mathematically by carefully defining and limiting the problem. The simplified instruction sequence problem analyzed in this appendix is overly constrained, but approximates actual instruction well enough that the results are qualitatively correct, and provide a lower bound on the number of instruction sequences possible in the general case.

The analysis shows that the number of legal instruction sequences that can produce a given sequence of physical operations grows very quickly with the number of physical operations. Disregarding abstract instructions, it is found that for only six physical operations, over 100 instruction sequences are possible; for seven, the number is over 400. Allowing abstract instructions adds another exponential factor.

The results indicate the kind of flexibility required of an agent purporting to learn from tutorial instruction. Previous instructable agents have largely avoided the problem of learning from variable instruction sequences, either because they do not learn procedures from sequences of instruction (e.g., SHRDLU [Winograd, 1972], HOMER [Vere and Bickmore, 1990]), or because they require that instructions be given in the canonical execution order (e.g., using a robotic teach pendant [Segre, 1987]).

## B.1 Flexibility in procedural instruction

Consider being taught to change a tire. Say the desired sequence of physical operations is `<open-trunk, retrieve-jack, place-jack-under-frame,...>`. This task might be taught in a number of ways. For instance, the teacher might begin by saying "First, jack up the car." This does not correspond to a single primitive operation, but rather is a macro-operator (that the student

may or may not know). Or, the teacher might say "Place the jack under the frame of the car." assuming that you can infer the first two operations. Or, the instruction sequence could be started even further along the actual physical operation sequence: "First, you need to remove the bolts." Here, the operation is abstract (since you can't remove all the bolts at once, it doesn't correspond to a single physical operation), and the instructor has assumed (or plans to teach you later) that you know you need to jack up the car first.

This example illustrates two reasons why an instructor may not teach a task by simply indicating the exact sequence of primitive operators to be carried out:

1. The instructor might indicate an instruction that is applicable in a future state, assuming that the student knows how to reach that future state from the current state. If the instructor's assumption is incorrect, further instruction may be required. For example, if when giving you directions to another office I say "Go down to the end of the hall, turn left, ..." I am assuming that you know how to get *to* the hall. If you don't, I will need to tell you how (maybe, "Go through that doorway"). Thus, the instructions are "out of order": I told you to go through the doorway *after* I told you to go down the hall, but when executing the instructions, you will go through the doorway first.

2. The instructor may indicate an abstract operation or macro-operator that is not a primitive operation for the student. This could occur because the instructor simply does not know what operations the student knows, or because the instructor wants to teach the student a useful hierarchy of operators that can be used for later tasks (e.g., **remove-all-bolts** may be a useful macro).

## B.2 Formalized Problem Definition

Consider a situation in which an agent is given instructions to reach a pre-specified goal. The goal is reached by composing a set of known operators, each with pre- and post-conditions. The instructor guides the system to the goal by indicating an operator to perform next. To begin, the problem is limited to the case where each instruction directly corresponds to a single primitive operator. However, the indicated operator may not be directly performable (its preconditions may not be met). If it is not, the instructor must tell the system how to reach a state in which the chosen operator can be performed.

A simplified version of the problem can be formalized as follows. Call the available operators $o_1, o_2, o_3, \ldots, o_N$, and the world states **s(0),s(1),s(2),...s(N)**. Each operator $o_i$ has the precondition that it is applicable from state **s(i-1)**, and transforms the state to be **s(i)**. The problem is to get to state **s(G)**, starting from state **s(0)**.

Clearly, this problem is simplified in many ways from that of a full-blown agent. The states follow a linear sequence and are uniquely transformed by each operator. In a real agent, different operators can cause states to be transformed in a variety of ways. A real agent's operators will change only a portion of the state; thus an operator's applicability can be generalized to consider only the relevant portion. These possibilities are ignored in this analysis.

In addition, the problem as stated precludes higher level macro-operators that have no direct implementation (do not directly transform the state), but are implemented by sequences of lower level operators. These are considered below.

To better understand the type of operators being assumed for this analysis, consider again the example of changing a tire. The operators available for instruction might be things like **open-trunk**, **retrieve-jack**, **place-jack-under-frame**, **remove-bolt**, etc. **Remove-bolt** has the preconditions that the jack is under the frame, the car is raised, the hubcap is removed, and you are holding the wrench; its implementation is using the wrench to remove a bolt. Thus, each operator's implementation corresponds to the "last step" needed to achieve it.

One other constraint must be placed on the problem before the analysis can proceed. It is assumed that if the system is instructed to carry out an operator whose preconditions are not met, it creates a subgoal to reach a state where the preconditions are met. The instructor must then

$< o_1, o_2, o_3 >$ $< o_1, o_3, o_2 >$
$< o_2, o_1, o_3 >$ $< o_3, o_1, o_2 >$
$< o_3, o_2, o_1 >$

Figure B.1: Legal instruction sequences for reaching state `s(3)`.

instruct the system on how to achieve that subgoal, *before* continuing to instruct on how to reach the overall goal. For example, if the agent is in state `s(1)` and is instructed to perform operator $o_3$, the system will create a subgoal to reach state `s(2)`. The instructor must then indicate how this subgoal can be achieved (i.e., by performing $o_2$). This is the *one-goal-at-a-time* constraint: the instructions are constrained to apply to the most recent subgoal that has been formed.

## B.3 Analysis

An *instruction sequence* is a sequence $< o_i, o_j, o_k, \ldots >$ of operators such that the sequence allows `s(G)` to be reached while obeying the one-goal-at-a-time constraint mentioned above. That is, the instructor is required to instruct on how to complete the current subgoal before going on to other instruction for the overall goal. This means that *an operator may not be instructed until all operators with lower indices that have been previously instructed are completed.* For example, to reach `s(3)`, instruction sequence $< o_2, o_3, o_1 >$ is prohibited. Operator $o_2$ must be completed (requiring the selection of $o_1$) before $o_3$ can be chosen.

Define $f(G)$ to be the number of instruction sequences that will lead to goal `s(G)`. The initial state is `s(0)`; thus, to achieve `s(0)`, there is one possible instruction sequence (the empty sequence): $f(0) = 1$. To reach `s(1)`, the only possible sequence is $< o_1 >$, so $f(1) = 1$.

When the goal is `s(2)`, either a forwards or a backwards ordering is possible. That is, the instructions might be $< o_1, o_2 >$ (forwards), or $< o_2, o_1 >$ (backwards). In the latter case, when instruction $o_2$ is given, the system is in state `s(0)`, so $o_2$ cannot be executed. Therefore, a subgoal is formed to reach `s(1)`. The second instruction tells how this subgoal can be achieved; once it is, $o_2$ applies. This corresponds to the example in which a robot is told "Go down the hall," but does not know how to reach the hall, and so is next told, "Go through the doorway." Forwards and backwards orderings are the only possibilities, so $f(2) = 2$. Note that in both cases the *execution* sequence, that is, the order in which operations are actually executed, is 1 followed by 2.

For a goal of `s(3)`, the situation is more complex. The instructor may indicate either $o_1$, $o_2$, or $o_3$ as the initial operator to be achieved. If the initial instruction is $o_1$, the system moves to state `s(1)` and is left with the problem of reaching `s(3)` from there. This is exactly analogous to the problem of reaching `s(2)` starting from `s(0)`, and thus there are $f(2)$ instruction sequences to complete it (here, $< o_2, o_3 >$ and $< o_3, o_2 >$). If the initial instruction is $o_2$, a subgoal is formed to reach `s(1)`; there are $f(1)$ ways to do that (simply select $o_1$). This leaves the problem of reaching `s(3)` from `s(2)`, which is analogous to reaching `s(1)` from `s(0)`; there are, again, $f(1)$ ways to do that. Finally, if the initial instruction is $o_3$, this leaves the problem of reaching `s(2)` from `s(0)`, and there are $f(2)$ ways to do that. Thus:

$$f(3) = f(0) \cdot f(2) + f(1) \cdot f(1) + f(2) \cdot f(0) = 5$$

The set of five legal sequences is shown in Figure B.1.

The pattern seen here continues for higher goals. Denote the number of sequences possible to get from state `s(i)` to state `s(j)` as $F(i, j)$. It is clear that $F(i, j) = f(j - i)$. The goal `s(4)` has the following legal instruction sequences:

$$o_1 \quad \textit{followed by} \quad F(1,4) \textit{ sequences of } o_2, o_3, o_4$$

$< o_1, o_2, o_3, o_4 >$ $< o_1, o_2, o_4, o_3 >$
$< o_1, o_3, o_2, o_4 >$ $< o_1, o_4, o_2, o_3 >$
$< o_1, o_4, o_3, o_2 >$ $< o_2, o_1, o_3, o_4 >$
$< o_2, o_1, o_4, o_3 >$ $< o_3, o_1, o_2, o_4 >$
$< o_3, o_2, o_1, o_4 >$ $< o_4, o_1, o_2, o_3 >$
$< o_4, o_1, o_3, o_2 >$ $< o_4, o_2, o_1, o_3 >$
$< o_4, o_3, o_1, o_2 >$ $< o_4, o_3, o_2, o_1 >$

Figure B.2: Legal instruction sequences for reaching state **s(4)**.

$o_2$ *followed by* $o_1$ *and then* $F(2,4)$ *sequences of* $o_3, o_4$
$o_3$ *followed by* $F(0,2)$ *sequences of* $o_1, o_2$, *and then* $o_4$
$o_4$ *followed by* $F(0,3)$ *sequences of* $o_1, o_2, o_3$

This is:

$$f(4) = f(0) \cdot f(3) + f(1) \cdot f(2) + f(2) \cdot f(1) + f(3) \cdot f(0)$$

The set of 14 legal sequences is shown in Figure B.2.

It can be shown that this pattern continues for higher numbered goals. In general,

$$f(N) = f(0) \cdot f(N-1) + f(1) \cdot f(N-2) + f(2) \cdot f(N-3) + \cdots + f(N-1) \cdot f(0)$$

This recurrence relation produces a sequence known in mathematics as the Catalan numbers.[1] Catalan numbers count many interesting sets, such as the set of binary trees with $N+1$ leaves, and the number of grid paths from (0,0) to (N,N) in a cartesian coordinate system that stay below the diagonal line between those points. The closed form for Catalan numbers is:

$$f(n) = \frac{1}{n+1} \begin{pmatrix} 2n \\ n \end{pmatrix}$$

$f(5)$ is 42; $f(6)$ is 132; f(7) is 429.

## B.4 Adding Hierarchy

Considering only instructions that correspond directly to primitive operations, the analysis has shown that even a short sequence of primitive operations can be instructed by a large number of instruction sequences. Next, consider the possibility of instructions that impose hierarchy onto the set of primitive operations; that is, instructions that correspond to multiple basic actions.

There may or may not be a "natural" hierarchy to impose upon a given set of primitive operators. To be fully flexible, the agent must allow the instructor to impose any possible hierarchy onto the domain. In order to get an idea for how large the number of possible hierarchies might be, consider an example of imposing a hierarchical structure onto a sequence of primitive operations for lining up two blocks. Two possible hierarchies for the goal of lining up blocks are shown in Figure B.3.

The total number of hierarchies possible for a sequence of $N$ actions corresponds to the total number of (either partial or complete) trees that can be produced with $N$ leaves. The trees may be partial because the instructor may instruct some actions at the base level, without including them in a higher-level operation. The number of total trees with $N$ leaves grows at least exponentially in $N$ - it is easy to show, for instance, that $2^{N-1}$ is a lower bound.

[1] Thanks to Dale Darrow of the Penn State University mathematics department for identifying this recurrence relation.

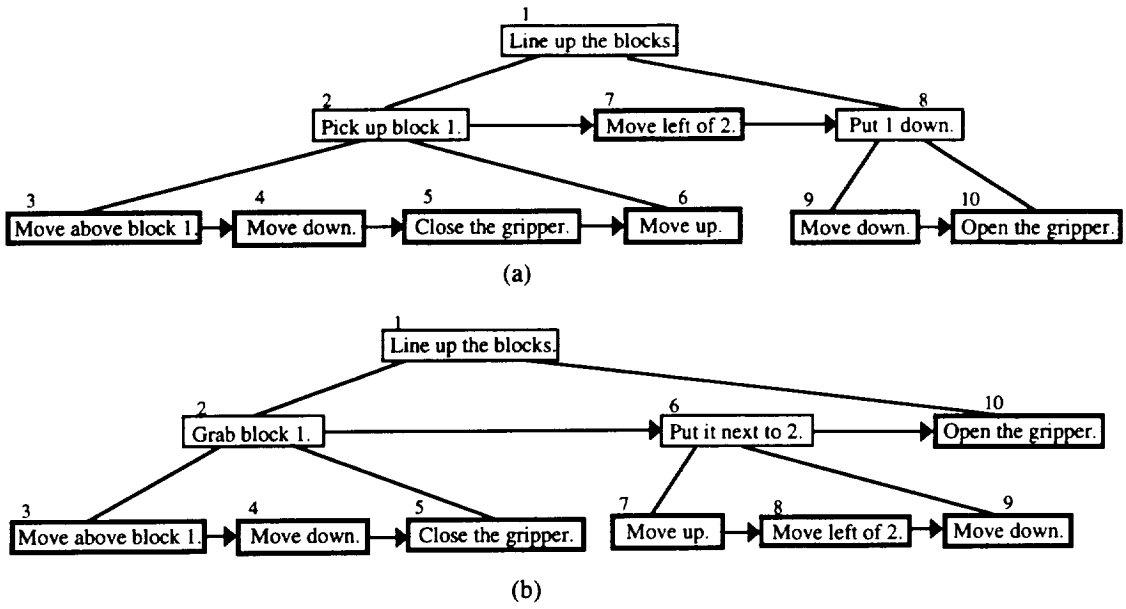



Figure B.3: Two possible instruction hierarchies for "Line up the blocks." Boldface boxes are primitive operations; numbers indicate one possible ordering of the instructions.

The point is simply that an instructor can impose any of a large (exponential) number of hierarchical structures onto a procedure, and an agent that learns from instruction must be able to learn from any of these (although some may be more difficult to learn than others).

## B.5 Conclusion

Although the analysis presented here is based on a simplified version of the problem, the point is clear: a given procedure may be taught in a great number of ways. Since the simplifications reduce the number of instruction sequences (by limiting the form of the states and operators), the results can be viewed as a lower bound. In real instruction the number of legal instruction sequences for a given procedure is even greater (although not all would be considered equally sensible).

The large number of possible instruction sequences indicates the amount of flexibility Instructo-Soar provides to its instructor. By allowing the instructor to skip steps and hierarchically structure procedures into subtasks in any way desired, Instructo-Soar can be taught a new procedure in any of a large number of different ways.

# Appendix C

# How chunking over internal projections produces general rules

This appendix describes in some detail how Instructo-Soar's internal forward projection of an instructed operator can result in learning general knowledge about the operator. In particular, it describes the example of learning a general rule for moving to a table to pick up a block, shown in Figure C.1, from the instructions for picking up a particular block. The overall example of learning the "pick up" procedure from instruction is described in Chapter 3.

## C.1 Chunking

General rules are created by Soar's learning mechanism, chunking, applied to the successful forward projection of an instruction. Chunking is a form of explanation-based learning [Mitchell *et al.*, 1986; DeJong and Mooney, 1986], but is organized somewhat differently than standard EBL methods like EBG [Mitchell *et al.*, 1986]. The relationship between chunking and standard EBL notions like training examples and goal concepts is described in more detail by Laird and Rosenbloom [1986] and by Ellman [1988].

In Soar, learning occurs whenever a result is returned from within a subgoal to resolve an impasse. The goal of the chunking process is to build a rule that will create the result of the subgoal based directly on conditions of the initial situation before the impasse occurred. When a similar situation occurs in the future, this rule will fire to produce the subgoal result directly, avoiding the impasse altogether.

A prototypical case is diagrammed in Figure C.2. The circled letters are properties that can be tested by rules. The arrows represent rule firings; the vertical dotted lines are the boundaries of a subgoal. Here, an impasse occurs when Soar is in the situation with properties A, B, C, and D; the impasse is resolved when resulting subgoal creates R. E,F, G, and H are intermediate results created within the subgoal.

**If** the goal (higher-level operator) is **new-op-14** with **object** **<obj>**, and
**<obj>** is sitting on a table **<table>**, and
the robot is not docked at **<table>**, and
**<obj>** is **small**, and
the gripper has **status open**,
**then** propose operator **move-to-table(<table>)**.

Figure C.1: The general operator proposal rule learned from the instruction "Move to the yellow table" (**new-op-14** is the newly learned "pick up" operator).

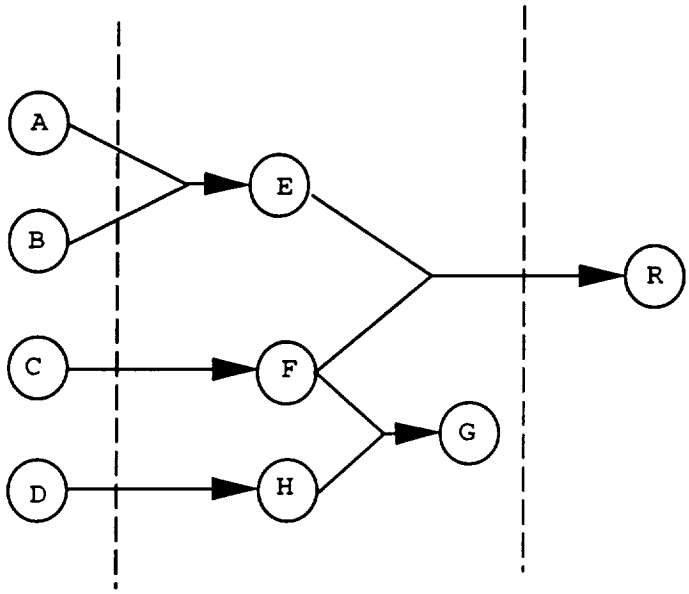



Figure C.2: An example of processing that leads to chunking.

When R is produced, chunking operates to create a chunk that will produce R directly from the initial situation in the future, avoiding all of the processing in the subgoal. A chunk is built through two steps:

1. **Back-tracing.** Chunking begins by determining which conditions of the initial situation led to the creation of the result. When the result is produced, chunking back-traces through the processing that caused its creation. The back-trace moves back through the rules that applied within the subgoal until it reaches the conditions of the initial situation that were tested. In the case of Figure C.2, the back-trace starts from R, moves back to E and F, and then to A, B, and C. Thus A, B, and C are the initial properties that led to R's creation, and chunking creates a rule of the form $A, B, C \rightarrow R$.

2. **Variablization.** Next, the rule $A, B, C \rightarrow R$ is made more general by converting certain identifiers within the rule into variables. Constant values that were directly tested (e.g., `color(red)`) are not variablized, but all other identifiers are (e.g., `pickup(b12)` becomes `pickup(<b>)`, where `<b>` is a variable).[1] Thus, the final result is a variablized form of the rule $A, B, C \rightarrow R$.

## C.2 Chunking over forward projection

This same process applies to a more complex situation to learn the general **move-to-table** chunk. The processing that causes the chunk to be learned – a situated explanation of the instruction "Move to the yellow table" by forward internal projection – is shown in Figure C.3. In the figure, the large arrowheads moving left-to-right represent impasses that lead to subgoals; the right-to-left arrowheads at the bottom represent resolution of those impasses by returning the indicated results. The agent begins in the state pictured, and selects **new-op14(blk-r)** (the operator for picking up the red block) to perform. An impasse arises because the agent does not know what sub-operator to select to begin carrying out **new-op-14**. In the resulting subgoal, the agent attempts to recall past instruction, and recalls "Move to the yellow table." Rather than immediately returning this as the next operator to perform, the agent first attempts to *evaluate* it (by selecting the operator **eval(move-to-table(tbl-y))**), to verify that it will be successful in the current situation. There

[1] The details of how Soar variablizes rules are given in [Laird *et al.*, 1993].

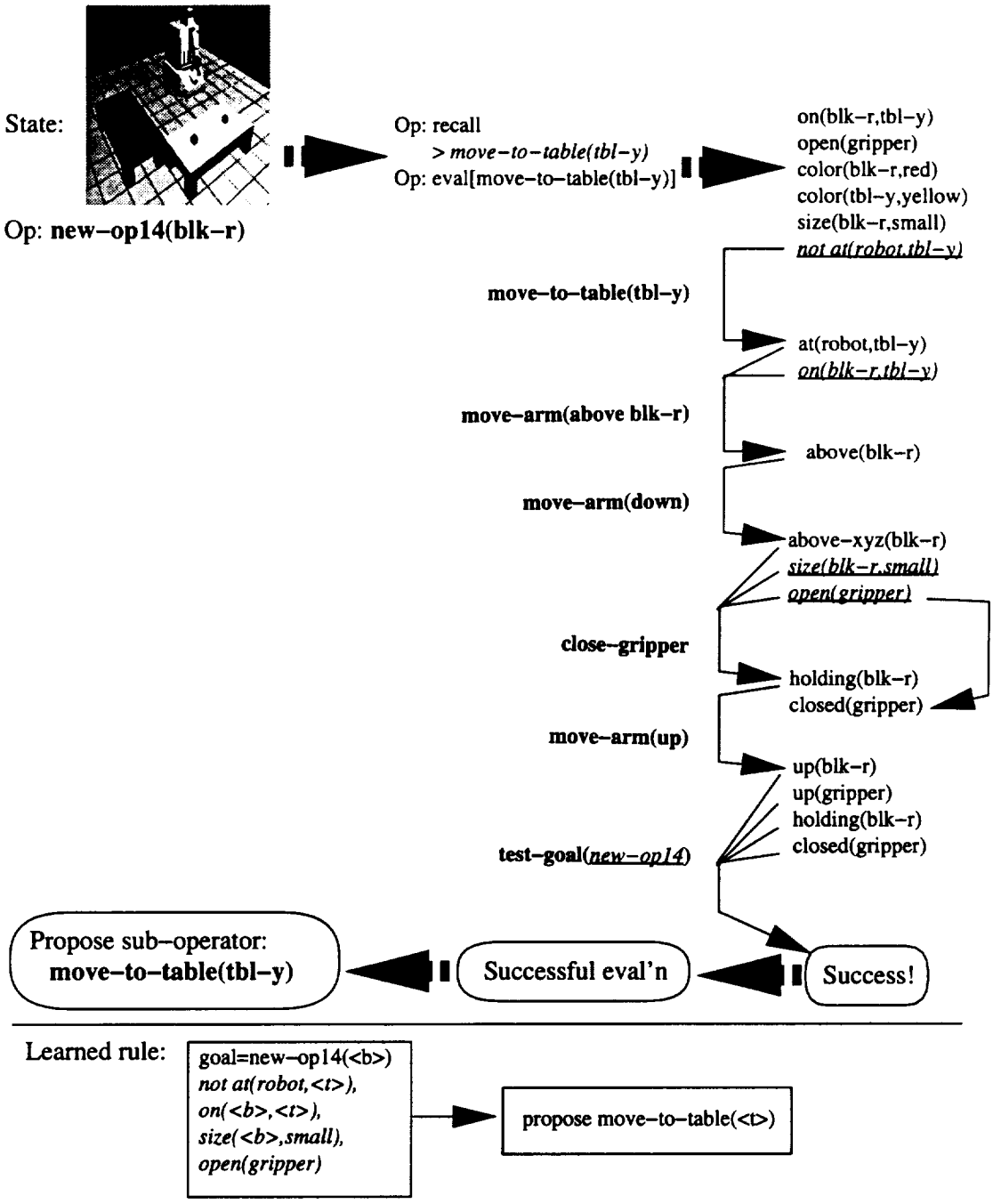



Figure C.3: How internally projecting an instruction leads to learning a general rule.

is no directly available knowledge indicating this evaluation, so an impasse arises. In the resulting subgoal (in the rightmost column of Figure C.3), an internal forward projection of the recalled operator is carried out, in order to verify its successful use in the current situation.

The top of the rightmost column shows the starting state for the forward projection (a duplicate of the pictured current state) as a set of properties. Starting from this state, the agent projects the effects of each operator in the execution sequence of **new-op14**, starting with the recalled **move-to-table** operator. The projection moves from the top to the bottom of the figure. Each arrow represents a piece of operator effects knowledge that the agent applies to move from one state to another. These operator effects rules indicate the dependencies between each property in the state after an operator is projected and the state it is projected from. Rather than reproduce every property of every state, after the initial state the figure only shows the properties that the projection depends on. The properties shown underlined in italics are the properties that hold in the initial state, that are tested by rules during the projection.

After forward projecting the final execution step of **new-op14** (moving the arm up), the operator **test-goal** is selected. This operator contains the knowledge of **new-op14**'s termination conditions; it tests that the four termination conditions shown hold, and because they do, it declares the forward projection a success. The goal of the projection has been successfully reached. Thus, a successful evaluation of **move-to-table(tbl-y)** is returned, to resolve the impasse of the **eval** operator. Because of this successful evaluation, the agent proposes the **move-to-table(tbl-y)** operator to perform, resolving the initial impasse.

As mentioned, in Soar learning occurs whenever a result is returned from within a subgoal to resolve an impasse. In this case, two results were returned from subgoals: the successful evaluation of **move-to-table** from the projection subgoal, and the proposal of **move-to-table** to resolve the initial impasse. Thus, two chunks are learned.

Consider chunking's performance to learn the operator proposal rule of Figure C.1, since it was the primary target of the instruction. Chunking back-traces from the proposal of the operator, back through the successful evaluation, and then through all of the rules that fired within the internal projection to cause the evaluation. The back-trace ends when it reaches conditions from the initial (pictured) state. The conditions found by this back-trace are those shown in underlined italics. These conditions come from the initial state by being copied down from the pictured state to the starting projection state (at the top of the rightmost column) and then tested within the projection to produce the successful evaluation. In addition to these state conditions, **new-op14(blk-r)** was also tested in the projection (to produce the **test-goal(new-op14)** operator).

After back-tracing gathers these conditions into a rule, the rule is variablized to produce the resulting chunk shown at the bottom of the figure. Identifiers for things like the specific block and table are variablized by chunking, while specific constants that were tested, like **small**, are not variablized.

The resulting rule proposes moving to the appropriate table when trying to pick up (**new-op14**) something. Back-tracing over the successful forward projection allowed identification of the key conditions under which moving to the table will be a successful action to take. Extraneous properties, like the colors of the block and table, were not included in the chunk because the success of the projection did not depend on them.

# Bibliography

[Akatsuka, 1986] Noriko Akatsuka. Conditionals are discourse-bound. In Elizabeth Closs Traugott, editor, *On Conditionals*, pages 333–51. Cambridge Univ. Press, Cambridge, 1986.

[Allen and Perrault, 1980] James F. Allen and C. Raymond Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178, 1980.

[Alterman and Carpenter, 1991] Richard Alterman and Tamitha Carpenter. Reading instructions. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 653–657, August 1991.

[Alterman and Zito-Wolf, 1993] Richard Alterman and Roland Zito-Wolf. Agents, habitats, and routine behavior. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 305–310, 1993.

[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.

[Anderson, 1983] John R. Anderson. *The architecture of cognition.* Harvard University Press, Cambridge, MA, 1983.

[Anderson, 1987] John R. Anderson. Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, 94(2):192–210, 1987.

[Annett, 1991] John Annett. Skill acquisition. In J. E. Morrison, editor, *Training for performance: Principles of applied human learning*, pages 13–51. John Wiley and Sons Ltd., 1991.

[Balkany *et al.*, 1991] Alan Balkany, William P. Birmingham, and Iris D. Tommelein. A knowledge-level analysis of several design tools. In *Artificial Intelligence in Design*, 1991.

[Birmingham and Klinker, 1993] William Birmingham and Georg Klinker. Knowledge acquisition tools with explicit problem-solving methods. *The Knowledge Engineering Review*, 8(1), 1993.

[Birmingham and Siewiorek, 1989] William P. Birmingham and Daniel P. Siewiorek. Automated knowledge acquisition for a computer hardware synthesis system. *Knowledge Acquisition*, 1:321–340, 1989.

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

[Bovair and Kieras, 1991] Susan Bovair and David E. Kieras. Toward a model of acquiring procedures from text. In Barr, Kamil, Rosenthal, and Pearson, editors, *Handbook of Reading Research, Volume II.* Longman, 1991.

[Bovair, 1991] Susan Jane Bovair. *A model of procedure acquisition from written instructions.* PhD thesis, The University of Michigan, Dept. of Psychology, 1991.

[Brachman, 1980] R.J. Brachman. An introduction to kl-one. In Brachman R. J., editor, *Research in Natural Language Understanding*, pages 13–46. Bolt, Beranek and Newman Inc., Cambridge, MA, 1980.

[Brown *et al.*, 1989] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational Researcher*, 18:32–42, 1989.

[Carbonell and Gil, 1987] Jaime G. Carbonell and Yolanda Gil. Learning by experimentation. In *Proceedings of the International Workshop on Machine Learning*, pages 256–265, 1987.

[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.

[Carroll, 1984] John M. Carroll. Minimalist training. *Datamation*, 30(18):125–136, 1984.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–29, 1986.

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

[Chi and VanLehn, 1991] M. T. H. Chi and K. VanLehn. The content of physics self-explanations. *Journal of the Learning Sciences*, 1(1):69–105, 1991.

[Chi *et al.*, 1989] M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13:145–182, 1989.

[Churchill and Young, 1991] Elizabeth F. Churchill and Richard M. Young. Modelling representations of device knowledge in soar. In Luc Steels and Barbara Smith, editors, *Artificial Intelligence and SImulation of Behavior*, pages 247–255. Springer-Verlag, 1991.

[Clancy, 1985] W. J. Clancy. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[Davis, 1979] R. Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12(2):409–427, 1979.

[DeJong and Mooney, 1986] Gerald F. DeJong and Raymond J. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[Dietterich, 1986] T. G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1:287–315, 1986.

[DiEugenio and Webber, 1992] B. DiEugenio and B. Webber. Plan recognition in understanding instructions. In J. Hendler, editor, *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 52–61, College Park, MD, 1992.

[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.

[Dixon *et al.*, 1988] P. Dixon, J. Faries, and G. Gabrys. The role of explicit action statements in understanding and using written directions. *Journal of Memory and Language*, 27:649–667, 1988.

[Dixon, 1982] Peter Dixon. Plans and written directions for complex tasks. *Journal of Verbal Learning and Verbal Behavior*, 21:70–84, 1982.

[Dixon, 1987] Peter Dixon. The processing of organizational and component step information in written directions. *Journal of Memory and Language*, 26:24–35, 1987.

[Drummond, 1989] Mark Drummond. Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation*, Toronto, Canada, May 1989. Morgan Kaufmann.

[Ellman, 1989] Thomas Ellman. Explanation-based learning: A survey of programs and perspectives. *Computing Surveys*, 21(2):163–221, 1989.

[Emihovich and Miller, 1988] Catherine Emihovich and Gloria E. Miller. Talking to the turtle: A discourse analysis of Logo instruction. *Discourse Processes*, 11:183–201, 1988.

[Eshelman *et al.*, 1987] L. Eshelman, D. Ehret, J. McDermott, and M. Tan. MOLE: A tenacious knowledge-acquisition tool. *International Journal of Man-Machine Studies*, 26(1):41–54, 1987.

[Feltz and Landers, 1983] D. L. Feltz and D. M. Landers. The effects of mental practice on motor skill learning and performance: A meta-analysis. *Journal of Sport Psychology*, 5:25–57, 1983.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[File and Jew, 1973] S. E. File and A. Jew. Syntax and the recall of instructions in a realistic situation. *British Journal of Psychology*, 64:65–70, 1973.

[Fisher, 1987] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[Flann and Dietterich, 1989] Nicholas S. Flann and Thomas G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

[Foltz *et al.*, 1988] P. E. Foltz, S. E. Davies, P. G. Polson, and D. E. Kieras. Transfer between menu systems. In *Proceedings of the CHI 1988 Conference on Human Factors in Computing Systems.* ACM, 1988.

[Ford and Thompson, 1986] Cecilia A. Ford and Sandra A. Thompson. Conditionals in discourse: A text-based study from English. In Elizabeth Closs Traugott, editor, *On Conditionals*, pages 353–72. Cambridge Univ. Press, Cambridge, 1986.

[Frederking, 1988] R. E. Frederking. *Integrated natural language dialogue: A computational model.* Kluwer Academic Press, Boston, 1988.

[Gershman, 1979] Anatole V. Gershman. *Knowledge-Based Parsing.* PhD thesis, Yale University, Department of Computer Science, 1979.

[Ginsberg, 1988] Allen Ginsberg. Theory revision via prior operationalization. In *Proceedings of the Seventh National Conference on Artifical Intelligence*, pages 590–595, 1988.

[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.

[Gordon and Subramanian, 1993] Diana F. Gordon and Devika Subramanian. A multistrategy learning scheme for assimilating advice in embedded agents. In *Proceedings of the Second International Workshop on Multistrategy Learning*, Harper's Ferry, WV, 1993.

[Gray and Orasanu, 1987] Wayne D. Gray and Judith M. Orasanu. Transfer of cognitive skills. In S. M. Cormier and J. D. Hagman, editors, *Transfer of learning: Contemporary research and applications*, pages 183–215. Academic Press, Inc., 1987.

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

[Grosz, 1980] Barbara J. Grosz. Focusing and description in natural language dialogues. In A. K. Joshi, I. A. Sag, and B. L. Webber, editors, *Elements of discourse understanding: Proceedings of a workshop on computational aspects of linguistic structure and discourse setting*, pages 84–105. Cambridge University Press, Cambridge, England, 1980.

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

[Guha and Lenat, 1990] R. V. Guha and D. B. Lenat. Cyc: A mid-term report. *AI Magazine*, 11(3):32–59, 1990.

[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.

[Haiman, 1978] John Haiman. Conditionals are topics. *Language*, 54:564–89, 1978.

[Hall, 1986] R.J. Hall. Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence*, pages 568–572, Philadelphia, PA, August 1986.

[Hanks, 1990] Steven Hanks. Practical temporal projection. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Mass., 1990. AAAI Press.

[Hayes-Roth *et al.*, 1981] Frederick Hayes-Roth, Philip Klahr, and David J. Mostow. Advice taking and knowledge refinement: An iterative view of skill acquisition. In John R. Anderson, editor, *Cognitive skills and their acquisition*, pages 231–253. Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

[Holland, 1986] John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach, Volume II.* Morgan Kaufmann, 1986.

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

[Huffman and Laird, 1993a] Scott B. Huffman and John E. Laird. Instructo-Soar: Learning procedures from interactive instruction (video abstract). In R. Fikes and W. Lehnert, editors, *Proceedings of the National Conference on Artificial Intelligence*, page 857, 1993. Video presented at AAAI-93.

[Huffman and Laird, 1993b] Scott B. Huffman and John E. Laird. Learning procedures from interactive natural language instructions. In P. Utgoff, editor, *Machine Learning: Proceedings of the Tenth International Conference*, 1993.

[Huffman and Laird, 1994] Scott B. Huffman and John E. Laird. Acquiring procedures from tutorial instruction. In B. Gaines, editor, *Proceedings of the 1994 Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1994.

[Huffman *et al.*, 1993a] Scott B. Huffman, Craig S. Miller, and John E. Laird. Learning from instruction: A knowledge-level capability within a unified theory of cognition. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 114–119, 1993.

[Huffman *et al.*, 1993b] Scott B. Huffman, Douglas J. Pearson, and John E. Laird. Correcting imperfect domain theories: A knowledge-level analysis. In Susan Chipman and Alan L. Meyrowitz, editors, *Foundations of Knowledge Acquisition: Cognitive Models of Complex Learning*. Kluwer Academic, 1993. Also available as technical report number CSE-TR-114-91, University of Michigan Department of Electrical Engineering and Computer Science, November 1991.

[Huffman, 1992] Scott B. Huffman. A problem space and symbol level description of instructo-soar. Artificial Intelligence Laboratory, University of Michigan., 1992.

[Johnson-Laird, 1986] P. N. Johnson-Laird. Conditionals and mental models. In Elizabeth Closs Traugott, editor, *On Conditionals*. Cambridge Univ. Press, Cambridge, 1986.

[Johnson, 1982] P. Johnson. The functional equivalence of imagery and movement. *Quarterly Journal of Experimental Psychology*, 34A:349–365, 1982.

[Jones, 1966] S. Jones. The effect of a negative qualifier in an instruction. *Journal of Verbal Learning and Verbal Behavior*, 5:497–501, 1966.

[Judd, 1908] C. H. Judd. The relation of special training and intelligence. *Educational Review*, 36:28–42, 1908.

[Just and Carpenter, 1976] Marcel A. Just and Patricia A. Carpenter. Verbal comprehension in instructional situations. In David Klahr, editor, *Cognition and Instruction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1976.

[Kalita and Badler, 1990] Jugal K. Kalita and Norman I. Badler. A semantic analysis of action verbs based on physical primitives. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 412–419, 1990.

[Kalita, 1991] Jugal K. Kalita. *Natural language control of animation of task performance in a physical domain*. PhD thesis, Univ. of Pennsylvania, Dept. of Computer and Information Science, June 1991.

[Kautz and Allen, 1986] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 32–37, 1986.

[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.

[Kieras and Bovair, 1986] David E. Kieras and Susan Bovair. The acquisition of procedures from text: A production-system analysis of transfer of training. *Journal of Memory and Language*, 25:507–524, 1986.

[Kintsch, 1986] Walter Kintsch. Learing from text. *Cognition and Instruction*, 3(2):87–108, 1986.

[Kodratoff and Tecuci, 1987a] 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.

[Kodratoff and Tecuci, 1987b] Yves Kodratoff and Gheorghe Tecuci. Techniques of design and DISCIPLE learning apprentice. *International Journal of Expert Systems*, 1(1):39–66, 1987.

[Konoske and Ellis, 1986] P. J. Konoske and E. G. Ellis. Cognitive factors in learning and retention of procedural tasks. Technical Report NPRDC 87-14, Navy Personnel Research and Development Center, San Diego, CA, 1986.

[Laird and Rosenbloom, 1990] John E. Laird and Paul S. Rosenbloom. Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1022–1029, Boston, Mass., 1990. AAAI Press.

[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.*, 1989] John E. Laird, Eric S. Yager, Christopher M. Tuck, and Michael Hucka. Learning in tele-autonomous systems using Soar. In *Proceedings of the NASA Conference on Space Telerobotics*, 1989.

[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.

[Laird *et al.*, 1993] John E. Laird, Clare Bates Congdon, Erik Altmann, and Robert Doorenbos. Soar user's manual, version 6, 1993.

[Laird, 1983] John E. Laird. *Universal Subgoaling.* PhD thesis, Computer Science Department, Carnegie-Mellon University, 1983.

[Laird, 1988] John E. Laird. Recovery from incorrect knowledge in Soar. In *Proceedings of the National Conference on Artificial Intelligence*, pages 618–623, August 1988.

[Lave and Wenger, 1991] Jean Lave and Etienne Wenger. *Situated Learning: Legitimate peripheral participation.* Cambridge Univ. Press, Cambridge, 1991.

[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.

[Lehman *et al.*, 1993] Jill Fain Lehman, Allen Newell, Thad Polk, and Richard Lewis. The role of language in cognition: A computational inquiry. In *Conceptions of the Human Mind.* Lawrence Erlbaum Associates, Inc, 1993.

[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.

[Lewis, 1988] Clayton Lewis. Why and how to learn why: Analysis-based generalization of procedures. *Cognitive Science*, 12:211–256, 1988.

[Lewis, 1993] Richard L. Lewis. *An Architecturally-Based Theory of Human Sentence Comprehension.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1993. Forthcoming.

[Lindsay, 1963] Robert K. Lindsay. Inferential memory as the basis of machines which understand natural language. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 217–233. R. Oldenbourg KG., 1963.

[Lintern, 1991] Gavan Lintern. Instructional strategies. In J. E. Morrison, editor, *Training for performance: Principles of applied human learning*, pages 167–191. John Wiley and Sons Ltd., 1991.

[Litman and Allen, 1987] Diane J. Litman and James F. Allen. A plan recognition model for subdialogues in conversations. *Cognitive Science*, 11:163–200, 1987.

[Litman and Allen, 1990] Diane J. Litman and James F. Allen. Discourse processing and commonsense plans. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intention in Communication*, pages 365–388. MIT Press, 1990.

[Luchins, 1942] A. Luchins. Mechanization of problem solving. *Psychological Monographs*, 54(248), 1942.

[Mann and Thompson, 1988] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[Marcus and McDermott, 1989] S. Marcus and J. McDermott. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37, 1989.

[Markovitch and Scott, 1988] Shaul Markovitch and Paul D. Scott. The role of forgetting in learning. In J. Laird, editor, *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, 1988.

[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.

[McDermott, 1988] J. McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Knowledge Based Systems*, pages 120–146. Kluwer Academic, 1988.

[Michalski and Stepp, 1983] R. S. Michalski and Robert E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning - An Artificial Intelligence Approach*. Tioga, Palo Alto, CA, 1983.

[Miller and Johnson-Laird, 1976] George A. Miller and Philip N. Johnson-Laird. *Language and Perception*. Cambridge Univ. Press, Cambridge, 1976.

[Miller, 1993] Craig M. Miller. *A model of concept acquisition in the context of a unified theory of cognition*. PhD thesis, The University of Michigan, Dept. of Computer Science and Electrical Engineering, 1993.

[Minton *et al.*, 1989] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem-solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[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.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[Mohammed and Swales, 1984] M. A. H. Mohammed and J. M. Swales. Factors affecting the successful reading of technical instructions. *Reading in a Foreign Language*, 2:206–217, 1984.

[Mooney, 1990] Raymond J. Mooney. Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science*, 14:483–509, 1990.

[Mostow, 1981] David J. Mostow. *Mechanical transformation of task heurisitics into operational procedures*. PhD thesis, Carnegie-Mellon University, Department of Computer Science, April 1981.

[Mostow, 1983a] D. J. Mostow. Learning by being told: Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach*. Morgan Kaufmann, 1983.

[Mostow, 1983b] Jack Mostow. A problem-solver for making advice operational. In *Proceedings of the National Conference on Artificial Intelligence*, pages 279–283, 1983.

[Musen, 1989] M. A. Musen. Automated support for building and extending expert models. *Machine Learning*, 4(3-4):347–376, 1989.

[Newell and Simon, 1972] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice Hall, Englewood Cliffs, N.J., 1972.

[Newell *et al.*, 1990] Allen Newell, Gregg Yost, John E. Laird, Paul S. Rosenbloom, and Erik Altmann. Formulating the problem space computational model. In *Proceedings of the 25th Anniversary Symposium, School of Computer Science, Carnegie Mellon University*, September 1990.

[Newell, 1973] Allen Newell. You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase, editor, *Visual Information Processing*. Academic Press, New York, 1973.

[Newell, 1980] Allen Newell. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson, editor, *Attention and Performance VIII*. Erlbaum Associates, Hillsdale, N.J., 1980.

[Newell, 1981] Allen Newell. The knowledge level. *AI Magazine*, 2(2):1–20, 1981.

[Newell, 1990] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[Ourston and Mooney, 1990] Dirk Ourston and Raymond J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the National Conference on Artificial Intelligence*, pages 815–820, 1990.

[Palinscar, 1986] A. Palinscar. The role of dialogue in providing scaffolded instruction. *Educational Psychologist*, 21:73–98, 1986.

[Pazzani, 1991a] Michael Pazzani. A computational theory of learning causal relationships. *Cognitive Science*, 15:401–424, 1991.

[Pazzani, 1991b] 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. A symbolic solution to intelligent real-time control. *IEEE Robotics and Autonomous Systems*, 1993. In press.

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

[Porter *et al.*, 1990] B. W. Porter, R. Bareiss, and R. C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(3):229–263, 1990.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Redmond, 1992] Michael A. Redmond. *Learning by observing and understanding expert problem solving*. PhD thesis, Georgia Institute of Technology, 1992.

[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 Laird, 1986] Paul S. Rosenbloom and John E. Laird. Mapping explanation-based generalization onto Soar. In *Proceedings of the National Conference on Artificial Intelligence*, pages 561–567, August 1986.

[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.

[Rosenbloom *et al.*, 1988] Paul S. Rosenbloom, John E. Laird, and Allen Newell. The chunking of skill and knowledge. In H. Bouma and A. G. Elsendoorn, editors, *Working Models of Human Perception*, pages 391–410. Academic Press, London, England, 1988.

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

[Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland, editors. *Parallel distributed processing: Explorations in the microstructure of cognition.* MIT Press, Cambridge, MA, 1986.

[Rychener, 1983] Michael D. Rychener. The instructible production system: A retrospective analysis. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach*, pages 429–460. Morgan Kaufmann, 1983.

[Salganicoff, 1993] Marcos Salganicoff. Density-adaptive learning and forgetting. In P. Utgoff, editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283, Amheart, Mass., 1993.

[Sandberg and Wielinga, 1991] Jacobijn Sandberg and Bob Wielinga. How situated is cognition? In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 341–346, 1991.

[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.

[Schank, 1975] Roger C. Schank. *Conceptual Information Processing.* American Elsevier, New York, 1975.

[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.

[Shavlik *et al.*, 1991] Jude W. Shavlik, Raymond J. Mooney, and Geoffrey G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–144, 1991.

[Shen and Simon, 1989] Wei-Min Shen and Herbert A. Simon. Rule creation and rule learning through environmental exploration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 675–680, Detroit, Michigan, 1989.

[Simon and Hayes, 1976] Herbert A. Simon and John R. Hayes. Understanding complex task instructions. In David Klahr, editor, *Cognition and Instruction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1976.

[Simon, 1977] Herbert A. Simon. Artificial intelligence systems that understand. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1059–1073, Cambridge, Mass., 1977.

[Singley and Anderson, 1989] Mark K. Singley and John R. Anderson. *The transfer of cognitive skill.* Harvard University Press, 1989.

[Smith and Goodman, 1984] E. E. Smith and L. Goodman. Understanding written instructions: The role of an explanatory schema. *Cognition and Instruction*, 1:359–396, 1984.

[Sutton and Pinette, 1985] R. S. Sutton and B. Pinette. The learning of world models by connectionist networks. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 54–64, 1985.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Talmy, 1985] Leonard Talmy. Lexicalization patterns: semantic structure in lexical forms. In Timothy Shopen, editor, *Language typology and syntactic description*. Cambridge Univ. Press, 1985.

[Thorndike, 1903] E. L. Thorndike. *Educational Psychology*. Lemke and Buechner, New York, 1903.

[Thrun and Mitchell, 1993] S. B. Thrun and T. M. Mitchell. Integrating inductive neural network learning and explanation-based learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 930–936, 1993.

[Towell *et al.*, 1990] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the National Conference on Artificial Intelligence*, pages 861–866, 1990.

[VanLehn and Jones, 1991] K. VanLehn and R. Jones. Learning physics via explanation-based learning of correctness and analogical search control. In *Proceedings of the International Machine Learning Workshop*, 1991.

[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.

[VanLehn, 1990] K. VanLehn. *Mind bugs: The origins of procedural misconceptions*. MIT Press, Cambridge, Mass., 1990.

[Vera *et al.*, 1993] Alonzo H. Vera, Richard L. Lewis, and F. Javier Lerch. Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1993.

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

[Wertsch, 1979] James V. Wertsch. From social interaction to higher psychological processes: A clarification and application of Vygotsky's theory. *Human Development*, 22:1–22, 1979.

[Wilkins, 1990] David C. Wilkins. Knowledge base refinement as improving an incomplete and incorrect domain theory. In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 493–514. Morgan Kaufmann, 1990.

[Wilks, 1975] Y.A. Wilks. A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, 6, 1975.

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

[Wood *et al.*, 1976] David Wood, Jerome S. Bruner, and Gail Ross. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17:89–100, 1976.

[Wright and Wilcox, 1979] Patricia Wright and Penelope Wilcox. When two no's nearly make a yes: A study of conditional imperatives. In P. A. Kolers, M. E. Wrolstad, and H. Bouma, editors, *Processing of visible language*. Plenum, New York, 1979.

[Yost and Newell, 1989] Gregg R. Yost and Allen Newell. A problem space approach to expert system specification. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 621–7, 1989.

[Yost, 1993] Gregg R. Yost. Acquiring knowledge in Soar. *IEEE Expert*, 8(3):26–34, 1993.