

# Situated Plan Attribution<sup>1</sup>

**Randall W. Hill, Jr.**

Jet Propulsion Laboratory / Caltech  
4800 Oak Grove Drive M/S 525-3631  
Pasadena, CA 91109-8099  
hill@negev.jpl.nasa.gov

**W. Lewis Johnson**

USC / Information Sciences Institute  
4676 Admiralty Way  
Marina del Rey, CA 90292-6695  
johnson@isi.edu

To appear in the *Journal of Artificial Intelligence in Education*.  
Copyright © 1995, Association for the Advancement of Computing in Education.

## Abstract

Plan recognition techniques frequently make rigid assumptions about the student's plans, and invest substantial effort to infer unobservable properties of the student. The pedagogical benefits of plan recognition analysis are not always obvious. We claim that these difficulties can be overcome if greater attention is paid to the situational context of the student's activity and the pedagogical tasks which plan recognition is intended to support. This paper describes an approach to plan recognition called *situated plan attribution* that takes these factors into account. It devotes varying amounts of effort to the interpretation process, focusing the greatest effort on interpreting *impasse points*, i.e., points where the student encounters some difficulty completing the task. This approach has been implemented and evaluated in the context of the REACT tutor, a trainer for operators of deep space communications stations.

## Introduction

Plan recognition and agent modeling capabilities are valuable for intelligent tutoring (Corbett et al., 1990; Johnson, 1986), as well as other areas such as natural language processing (Charniak & Goldman, 1991), expert consultation (Calistri, 1990), and tactical decision making (Azarewicz et al., 1986). However, such capabilities are difficult to implement and employ effectively, for the following reasons. Plan recognition techniques can be *rigid*--they assume the agent is following a known plan step by step, and have difficulty interpreting deviations from the plan. The modeling process can be *underconstrained*, postulating mental activities that are difficult to infer from the agent's observable actions. An example of this style of modeling can be seen in (Ward, 1991), where the tutor attempts to track the student by generating production paths that could have led to an observed action. Finally, they tend to be *unfocused*--they do not target their analysis on those situations where tutorial intervention is warranted. For instance, intelligent tutors that use model tracing (Anderson et al., 1990) to interpret student actions tend to intervene whenever the student wanders off of a correct solution path; this intervention policy is potentially

---

<sup>1</sup>Portions of this paper are based on the AAI-94 paper entitled "Situated Plan Attribution for Intelligent Tutoring Systems." (Hill & Johnson, 1994)

disruptive and does not appear to be based on an analysis of whether it is appropriate to intervene.

This paper describes an approach to plan recognition called *situated plan attribution* that takes these factors into account. Situated plan attribution analyzes both the student's actions and the environmental situation. Attention to the situation is important because it allows the plan recognizer to recognize when the student must deviate from the usual plan, as well as alternative ways of achieving the goals of the plan. This flexibility avoids the rigidity problems of other techniques such as Kautz and Allen's deductive approach (Kautz & Allen, 1986), which assumes that all possible ways of performing an action are known, and every action is a step in a known plan. The ability to mix goal-directed and reactive behavior has also been found to be important for situated agents which must model the intentions of other agents (Tambe & Rosenbloom, 1994).

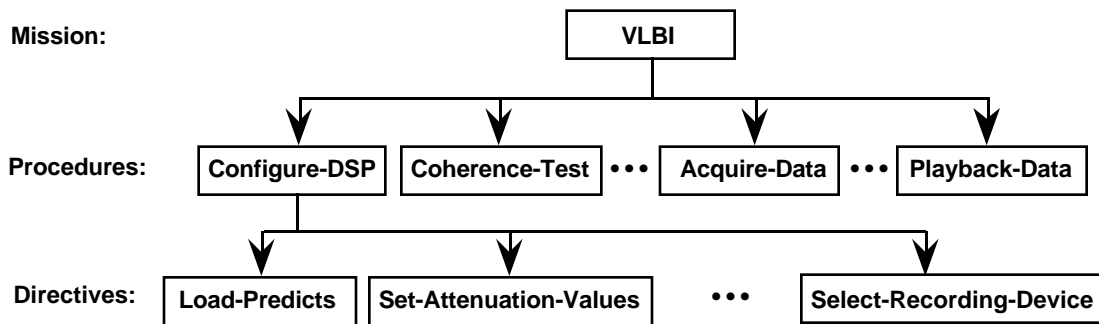
We will describe a tutor that implements this approach for a situated training task. It incorporates components for tracing plans, monitoring the environment, adapting plans, and explaining to students how to deal with problematic situations and plan failures. It employs a machine learning mechanism called chunking, implemented in the Soar cognitive architecture, which enables the tutor incrementally to combine these various types of knowledge into rules that suggest tutorial responses to student actions and situational events. The tutor is thus able to improve its own performance over time.

### **Context: Operator Training**

The situated plan attribution approach was developed and implemented in order to construct an intelligent tutoring system called REACT for training human operators of complex devices. The training domain is the operation of a communications link in NASA's Deep Space Network (DSN). The DSN is a worldwide system for navigating, tracking and communicating with all of NASA's unmanned interplanetary spacecraft. Operators of the DSN are responsible for initializing and controlling a complex array of devices, which range from the hydraulic pumps used to move a 70-meter antenna to the software that controls the receivers, exciters and digital spectral processor (DSP). They must be able to carry out complex procedures accurately, as well as recognize and respond to unexpected situations when they arise.

Tasks in this domain are organized into three levels: *mission*, *procedure*, and *directive*. The mission is a description of the overall task: it has a set of goals, it has a collection of devices assigned to a communications link, and it has a set of procedures, where some of the procedures may be common to other mission types. Procedures also have goals, usually with respect to the state of the devices that they affect. Each procedure has a sequence of di-

rectives that will, under ideal circumstances, cause the link devices to behave in a desired manner. A directive is a command that is issued by the link operator to control a device in the communications link. For each directive issued, a directive response is sent back indicating whether the device accepted or rejected the directive. If the directive is accepted, the operator watches for event notice messages and attends to subsystem displays for indications that the directive had its intended effect. Rejected directives must be re-sent by the operator since they have no effect on the state of the system whatsoever until they are accepted.



**Figure 1** Example of a task organized into three levels: Mission, Procedure, Directive. The cognitive model treats the task organization as a problem space hierarchy, where each box represents a problem space.

An example of a task in this domain is shown in Figure 1. The mission is VLBI (Very Long Baseline Interferometry), and it involves performing the procedures called Configure-DSP, Coherence-Test, and so on. The Configure-DSP procedure has directives to: load the mission-specific prediction data file (Load-Predicts), set the attenuation values on the Intermediate Frequency Video Down Converter (Set-Attenuation-Values), ..., and select a recording device to capture the mission's communication data (Select-Recording-Device). Each of the directive actions involves issuing a command (e.g., the Load-Predicts directive is `NLOAD predicts-file`).

Looking at Figure 1, it would appear that the tasks in this domain are straightforward and would require little or no training -- just follow the mission procedure manuals. We have found, however, that this is not the approach taken by domain experts. Through interviews with expert operators and system engineers, we determined that the procedure manuals only provide a subset of the knowledge needed to successfully perform a mission task. What is generally lacking in the procedure manuals is a complete description of the required device state conditions before and after a directive is issued. An expert operator knows the preconditions and postconditions for each directive and verifies these conditions are satisfied before and after each directive is sent. Operators who lack this knowledge may find it difficult to complete even simple procedures, since the directives may be rejected, or worse, put the device into an incorrect

state for the procedure or mission. For example, one of the preconditions for the Load-Predicts directive is that the predicts-file being loaded must be present on the system. If the Load-Predicts directive is issued for the predicts-file named "JK" (i.e., NLOAD JK), it will be rejected if a file by that name is not present in the predicts file directory.

Devices may enter unexpected states due to failures. Furthermore, commands may have unexpected effects when issued in the wrong situation. In either case, to become an expert operator requires learning to recognize the device state requirements (i.e., preconditions and postconditions) for each directive in every procedure. It also requires understanding the goals associated with each procedure, since in many instances it is necessary to work around device state anomalies in order to complete the procedure.

### **Impasse-Driven Tutoring**

Interactive simulations can be quite effective for training complex, situated tasks such as DSN operations. The REACT tutor is such a simulation-based training system. The student practices carrying out various missions by issuing directives to computer simulations of the real devices. They thereby become more facile with the mission procedures and gain experience in dealing with the various anomalous situations that may arise. The initial version of the REACT tutor incorporated custom-built simulations of the devices in the DSN communication link. Our current work makes use of the RIDES simulation authoring package (Munro et al., 1993) as a base.

Yet it was evident from the beginning that a simulation alone would be insufficient as a training system. It is not always evident to novices, or even to experts, why particular directives fail and are rejected. This can lead to confusion or frustration on the part of the student. Furthermore, it is not always apparent at the time whether a given mission was successfully completed. This is a serious problem for DSN operations: an incorrect configuration can cause the data being collected from the spacecraft to be garbled and meaningless, and this may not be discovered until weeks later when scientists inspect the data that was collected. Therefore, it is apparent that some form of on-line guidance or coaching is necessary in order to ensure that students learn effectively.

In order to determine how best to design a tutor for this task, a cognitive model of students interacting with simulated devices was constructed (Hill & Johnson, 1993a). One of the key conclusions from this study is that tutorial interaction should center around *impasse points*. An impasse is defined in this work to be an obstacle to problem solving that results from either a lack of knowledge or from incorrect knowledge (Brown & VanLehn, 1982; VanLehn, 1982, 1983; Hill, 1993). Our results agree with the results of earlier studies (e.g., VanLehn, 1988) that suggest that such impasse points are natural learning opportunities. When the student is at an

impasse, he or she naturally seeks information that can be used to overcome the impasse and continue the task. Information offered by the tutor at such points is readily accepted and assimilated.

These conclusions are the motivation for the approach to tutoring taken in the REACT system, called *impasse-driven tutoring*. In an impasse-driven tutoring system, the student is called upon to carry out specific tasks; in REACT's case, these tasks are missions performed with simulated devices. The tutor monitors the student's actions, but as long as he or she appears to be making progress, it does not intervene. If on the other hand the student appears to be at an impasse point, the tutor will offer suggestions as to how to resolve the impasse and proceed with the task. The tutor interrupts the students only when their actions are so faulty that it will be difficult or impossible for them ever to complete the task and there is no obvious cue from the simulation that things are going wrong.

In REACT's particular case, the student is assumed to have some understanding of operational procedures. However, the devices may be in unexpected states or behave in unexpected ways; the student must learn to recognize such situations and deviate from the standard procedures as necessary. REACT recognizes when the student has reached an impasse, because the student's action has failed or cannot achieve its intended purpose in the devices' current state. It then coaches the student through the impasse.

A tutor that is sensitive to such impasses does not run the risk of annoying the student with interruptions--the student's problem solving has already been interrupted by the impasse. The tutoring system need not intervene in a heavy-handed fashion; it can serve as an information resource that the student can turn to for assistance as needed. The student therefore has a greater sense of control over how the task is performed.

### **Situated Plan Attribution**

The question that is the focus of this paper is how best to track student performance in a situated activity such as device control, in support of impasse-driven tutoring. Previous approaches to student tracking have serious deficiencies in this context. Some do not provide the flexibility needed for such dynamic domains. Others incur substantial computational and development costs to generate information about the student that is of little value for the tutor. The situated plan attribution approach implemented in REACT avoids both sets of problems. It is highly flexible, able to cope easily with deviations from normal mission procedures. It is specifically designed to identify potential impasse points, and to understand enough of the student's plan to be able to offer useful advice at those points. This means that much of the computational cost of typical plan recognition or student modeling techniques can be avoided. Our stance is thus

consistent with that of (Self, 1990), who argues that to make student modeling tractable one must focus on realistic, useful objectives.

Plan recognition approaches such as that of (Kautz & Allen, 1986) and (Vilain, 1990) are inappropriate for domains such as the DSN because they are *rigid*, i.e., they make strong assumptions about the task and the knowledge employed to perform it, and do not degrade gracefully when those assumptions are violated. Kautz and Allen, for example, assume that both the observer and the agent have complete and correct knowledge about possible plans. Vilain makes similar assumptions, and further assumes that the plan library is sufficiently free of ambiguities that it can be encoded as a context-free grammar. In the DSN domain there is a standard set of procedures, which can serve as the basis for a plan library. However, the plans do not always work, and the students do not always know when they can be expected to work. Furthermore, the set of possible procedures is quite large, so constructing a complete plan library would be difficult.

Eliminating the assumptions that lead to rigidity tends to increase the complexity of the knowledge base and the computational cost of the plan recognition algorithms. PROUST (Johnson, 1986) is tolerant of plan variations and novel plans, but it requires detailed knowledge of the kinds of plan deviations that may be expected. Pathfinder (Calistri, 1990) requires knowledge of the probabilities of various misconceptions. The computational costs of these systems increases when students deviate from standard plans, because the systems try to generate complete interpretations of the students' actions. Other systems such as that of (Murray, 1986) and (Allemang, 1990) rely on theorem proving techniques which are intrinsically expensive to apply.

As it turns out, complex plan recognition algorithms are largely superfluous for situated tasks. The above approaches assume that one must analyze the plan in order to determine whether or not it is faulty. But in domains such as the DSN one need simply monitor the execution of the plan and see whether or not it has the desired effect. If the directives issued by the student are rejected, then the plan must be in error, and analysis of why the rejection occurred can help pinpoint the error. If the plan executes successfully (success in this case is limited to meaning that all of the plan's directives were sent and none were rejected), and has the desired effect (i.e., the plan's goals were achieved), then it must have been correct; at any rate no tutorial intervention is warranted, because no impasse occurred. Therefore the situated plan attribution approach involves simultaneous monitoring of the student's actions and the simulated environment, without expensive plan analyses.

Model tracing systems such as that of (Anderson et al, 1990) encounter difficulties that are in many ways similar to those of plan recognition systems. In the model tracing approach, an executable model of the student's performance is constructed and used to simulate the mental

activities of the student that would account for an observed behavior. Student actions are traced by running the executable model until it produces the same actions that were observed in the student. To trace a student's behavior using this technique means that the executable model must contain productions that choose the same action as the student took in a particular situation. In cases where the student takes the wrong action, the model tracer must also contain mal-rules that would simulate the student's faulty mental model that resulted in the wrong action being taken. Such executable cognitive models can be quite detailed--in extreme cases such as the system of (Ward, 1991), internal cognitive processes such as perception are modeled as well. Techniques such as these are inherently *underconstrained*: they waste effort performing searches that are unproductive, either because they do not lead to a solution, or because the resulting solutions are more detailed than is really necessary for tutoring.

The situated plan attribution approach adopts some of the features of the model tracing approach, while avoiding the problems that increase computational cost. It tracks the student's plan at an abstract level, at the level of missions and high-level procedures. This corresponds to model tracing in the sense that the system follows what the student is doing on the basis of its knowledge of how to carry out the task. Unlike model tracing, however, situated plan attribution does not use an executable cognitive model for most of the student tracking. The system traces the student by recognizing actions that are consistent with the expected plan. Actions that the system does not recognize are ignored, unless they have an undesirable effect on the state of one or more devices. An executable cognitive model, based on the one mentioned in the previous section, is employed only when an impasse is detected and the plan is found to be inappropriate for the current state of the devices. Furthermore, the emphasis here is not on inferring errors in the student's knowledge but in determining what knowledge an expert would require in order to overcome the impasse. Thus, computational effort is limited only to what is required in order to help students resolve impasses. Furthermore, effort decreases over time, because whenever the system employs the expert cognitive model to analyze the situation, it remembers the results of the analysis for use in similar situations.

Besides being influenced by plan recognition and model tracing, the situated plan attribution approach is influenced by research in situated cognition (Brown, 1990; Clancey, 1993). Cognition occurs in the context of an evolving situation; people attend to the situation, and their actions are in part responses to the situation. Therefore tutoring systems should attend to the student's situation as well and direct less effort at trying to guess what is going on in the student's head. In keeping with the idea that knowledge is socially constructed (Wittgenstein, 1953) we model the problem solving process in terms similar to those used by practitioners, e.g., missions, procedures, and directives in the DSN domain. Plans play a role similar to that advocated by (Suchman, 1987): they orient action, but do not dictate action in detail. Procedures in the DSN

domain are plans in this sense. We refer to the plan-matching process as plan attribution rather than plan recognition in order to remind ourselves that matching plans is simply a means to make sense of the trainee's actions, rather than making a detailed claim about the trainee's internal mental state.

Note, however, that concern with cognition in a situated context has not led us to abandon intelligent tutoring as an objective. We remain convinced that an understanding of the trainee's intentions is essential in order to make sense of errors when they occur (Johnson, 1986). When an impasse occurs, it is appropriate to refer to such intentions, if that helps trainees to better understand the cause of the impasse. There is no contradiction in modeling behavior as being both goal-directed and responsive to the situation; indeed, we have found in other domains that intelligent behavior in complex environments requires a combination of goal-directed and reactive behavior (Firby, 1987; Georgeff & Lansky, 1987; Kaelbling, 1987; Maes, 1989; Tambe et al., 1995). The REACT tutor detects student impasses using situated plan attribution and generates a tutoring explanation using an executable cognitive model. Explaining impasses requires going beyond the limits of the terminology of the community of practice, since DSN Operators do not customarily talk about such impasses. However, our experience is that once we explain to the trainees what we mean by impasse, the concept poses no difficulty.

We estimate that there are many real-world skills where feedback from the environment can guide the problem solving process as in REACT. Intelligent tutoring systems tend to overlook the role of the environment because they are frequently applied to abstract domains such as geometry or subtraction. Even in these domains there may be useful environmental cues to exploit. For example, intelligent tutors for programming tend not to take advantage of feedback from actually running the student's program, although recent work such as GIL is making such feedback more readily available to the student (Reiser et al., 1991).

### **Example Problem**

To illustrate how REACT works we will now describe an example from our task domain. Students are assigned missions that involve activities such as configuring and calibrating a set of communications devices, establishing a link to a spacecraft, recording data from the spacecraft, and transferring the recorded data to a control center. These tasks involve sending commands asynchronously via a computer terminal over a local area network to the devices. Standard command sequences for each type of mission are defined by procedure manuals. The devices initially respond to each command with an indication of whether the command is accepted or rejected; if accepted, the devices require time to change state.

Figure 2 shows two procedures. The first procedure, Configure-DSP, is used to configure the DSP subsystem, which is used for spectrum processing. The steps mostly involve loading or

setting parameters and selecting devices. The second procedure, Coherence-Test, is used to test the continuity and coherence of the communications link; it is supposed to be executed after the Configure-DSP procedure has been completed. These two procedures are examples of what the operator is supposed to do, though we found in examining operator logs that they rarely performed the procedures in the sequence given in the manual. The steps in these procedures can be executed out of order as long as their preconditions are met prior to execution. Because of the strong role that the situation plays in performing a procedure, there are many cases where the sequence used by the student differs from the manual.

Configure-DSP		Coherence-Test	
Command	Description	Command	Description
NLOAD <i>x</i>	load-predicts-file	NPCG <i>x</i>	set NPCG mode
NRMED <i>x</i>	select-recorder	NRUN <i>x</i>	run NCB program
SAT <i>x</i>	S-band attenuation	NDTE <i>x</i>	enable DTE
NTOP <i>x y</i>	set temperature	NFFT <i>x</i>	enable NFFT
OFST <i>x</i>	set offset time		

**Figure 2:** Example procedures

We will walk through the example shown in Figure 3 to illustrate how REACT overcomes the impediments to plan recognition. Here a student begins with Configure-DSP's first command for loading the predicts file, NLOAD JK. Line 1 shows the NLOAD command, and line 2 shows the device's response, COMPLETED, indicating that the command was accepted. Everything is proceeding as predicted by the plan: the correct command was issued by the student and it was accepted by the device.

Things get a bit more complicated on lines 3 through 7. On line 3 the student issues the next command in the Configure-DSP plan, NRMED. This command follows the Configure-DSP plan exactly, but the situation actually requires a different action to be taken, LD0 E, (i.e., enable recorder LD0), which is why the command is rejected on line 4. REACT thus must recognize when deviations from the plan are warranted; it does this by first noting the rejection and reasoning about why the action was not appropriate. In this case REACT determines that the command was rejected due to an action constraint violation (i.e., an unsatisfied precondition) of the NRMED command. REACT explains its reasoning to the student about the violation on line 4 as well as deriving a way to resolve the difficulty. The difficulty is viewed as an impasse because it prevents the student from continuing with the procedure, and it suggests a gap in the student's knowledge--if he had a good grasp of the procedure, he would have known to check the state of recorder LD0 before selecting it. At line 7 the student issues the NRMED command a second time; the plan calls for it to be issued just once. The second occurrence of the command is determined to be appropriate given that the first attempt at this action failed.

#	Commands / Responses	REACT's Explanation
1	> NLOAD JK	
2	> COMPLETED.	
3	> NRMED LD0	
4	>REJECTED. LD0 DISABLED	The NRMED command failed because one of its preconditions was unsatisfied: LD0 should be in the ONLINE mode instead of the OFFLINE mode. To resolve the impasse, Issue the command: LD0 E, Then Issue the command: NRMED LD0
5	> LD0 E	
6	> COMPLETED. LD0: ONLN	
7	> NRMED LD0	
8	> COMPLETED.	
9	> SAT 55	
10	> COMPLETED.	
11	> NTOP 20.0 30.0	
12	> COMPLETED.	
13	> NPCG MAN	
14	> COMPLETED.	You started the Coherence-Test procedure before you finished the Configure-DSP procedure. Issue the Command: OFST <>
15	> OFST 2.7	
16	> COMPLETED.	
17	> NRUN COLD	
18	> COMPLETED.	You failed to achieve one of the goals of the Configure-DSP procedure: SAT = 12. Issue the command: NIDLE REC, Then Issue the command: SAT 12
19	> NIDLE REC	
20	> COMPLETED.	
21	> SAT 12	
22	> COMPLETED.	
23	> NRUN COLD	
24	> COMPLETED.	
25	> NDTE E	
26	> COMPLETED.	
27	> NFFT E	
28	> COMPLETED.	

**Figure 3:** An example of tutoring with REACT

The example next illustrates difficulties that arise when the student follows a plan but fails to achieve its goals. The commands and responses on lines 9 through 14 follow the Configure-DSP plan exactly and all of the commands are accepted by the device. However, the parameter value of the SAT (Set S-Band attenuation value) command, 55, will not achieve one of the procedure's goals, that the value should be 12 by the time of the procedure's completion. This goal is not explicitly stated in the procedure, rather, it is derivable from the mission support data provided to the student. If the student does not correct this setting, it will affect the quality of the

communications link with the spacecraft and of the data being recorded. The failure to achieve a goal is another type of impasse that can occur when a student is performing a task, indicating another type of knowledge gap in the student's skill set. REACT gives the student the opportunity to correct the error alone, but will intervene if not, before it is too late to correct it. When it detects the NRUN COLD (i.e., run NCB program) command on line 19, which belongs to the Coherence-Test plan, it initiates the interaction concerning the unsatisfied goal. In this case REACT also employs its expert cognitive model to analyze the cause of the impasse and determine a solution.

The final point made by the example centers on the actions listed on lines 15 through 18. On line 15 the student sends the NPCG MAN (i.e., set the NPCG device to manual mode) command, which is the first command in the Coherence-Test procedure, prior to finishing the Configure-DSP procedure, which has OFST (set the offset time) as its last command. This is a straightforward case of misordered plans, and REACT immediately alerts the student that a step was missed prior to starting the new procedure (see line 16). REACT recognizes this type of impasse as a plan dependency violation.

Before going on to explain how REACT works, it should be noted that it may appear to the reader that REACT could replace the human operator altogether since it has the executable cognitive model of an expert operator: e.g., it is capable of goal-driven behavior, it can execute plans, and it can reactively plan to deal with unexpected situations. The work of automating all of the functions that a DSN operator performs is much more difficult, however, than building a tutor for a subset of the training tasks. In the long term it may be possible to automate the DSN operator's functions, but until then there will be a need for the type of training described in this paper.

### **How REACT Works**

Three types of impasses were introduced in the above example: (a) action constraint impasses, where the student takes an action that is in the plan but which the situation does not warrant, (b) goal failure impasses, where the student completes a plan without having achieved its goals, and (c) plan dependency impasses, where the student executes a plan before successfully completing one of its required predecessors. We will now give the details of how REACT recognizes and resolves each of these types of impasses.

### **Soar Cognitive Architecture**

REACT is implemented in Soar, an integrated problem solving and learning architecture that implements a theory of human cognition (Laird et al., 1987; Newell, 1990; Rosenbloom & Newell, 1986). The Soar architecture embodies the concept of problem solving as a goal-

oriented activity involving the search for and application of operators to a state in order to attain some desired results. Tasks in Soar are represented and performed in problem spaces. A Soar problem space consists of a collection of operators and states. Search takes place in the hierarchy of problem spaces. Operators are proposed, selected, and applied to the current state; the resulting state changes may cause other operators in the problem space to be proposed, selected, and applied, which goes on until the goal of the problem space is achieved. Impasses occur in Soar when the problem solver stops making progress. To resolve an impasse, the Soar problem solver creates a subgoal and selects a different problem space where other operators are available for solving the problem. When subgoal problem solving is successful, the results are saved in new productions created by Soar's chunking mechanism, which also saves the conditions that led to the impasse in the first place. The next time the conditions occur the learned chunk will be applied instead of having to search for an operator in the goal hierarchy. Thus, the problem space hierarchy is searched via subgoaling, and learning occurs when a subgoal yields a result.

### **Knowledge Representation in REACT**

REACT models several other aspects of plans besides the component actions shown in Figure 2, as will be briefly described below. For each type of mission the temporal precedence relationships among the plans is modeled with a directed graph structure called a temporal dependency network (TDN) (Fayyad & Cooper, 1992). A plan has a name and three attributes: state, execution status and goal status. The state of a plan can be either active or inactive; a plan is considered to be active once all of its predecessors in the TDN have been successfully completed. It is inactive prior to being active, and it becomes inactive again once it has been successfully completed. A plan's execution status (incomplete or complete) is determined by whether all of its commands have been observed. Each plan's goal status is marked satisfied if all its goals have been satisfied, otherwise it is unsatisfied.

Plans have two entities associated with them: operators (commands) and goals. The operators for the plans named Configure-DSP and Coherence-Test are shown in Figure 2. Each operator has a set of preconditions. A precondition is a tuple representing a device state that must be true before it can be considered satisfied. Similarly, a plan goal is also a tuple that represents a device state. As will be seen in the following sections, an active plan's goals are individually monitored for satisfaction at all times.

### **Problem Solving Organization of REACT**

The problem solving in REACT is organized into two high-level activities: *plan tracking* and *impasse interpretation*. Plan tracking involves watching the student interact with the environment and deciding whether the student's actions are appropriate for the assigned task and

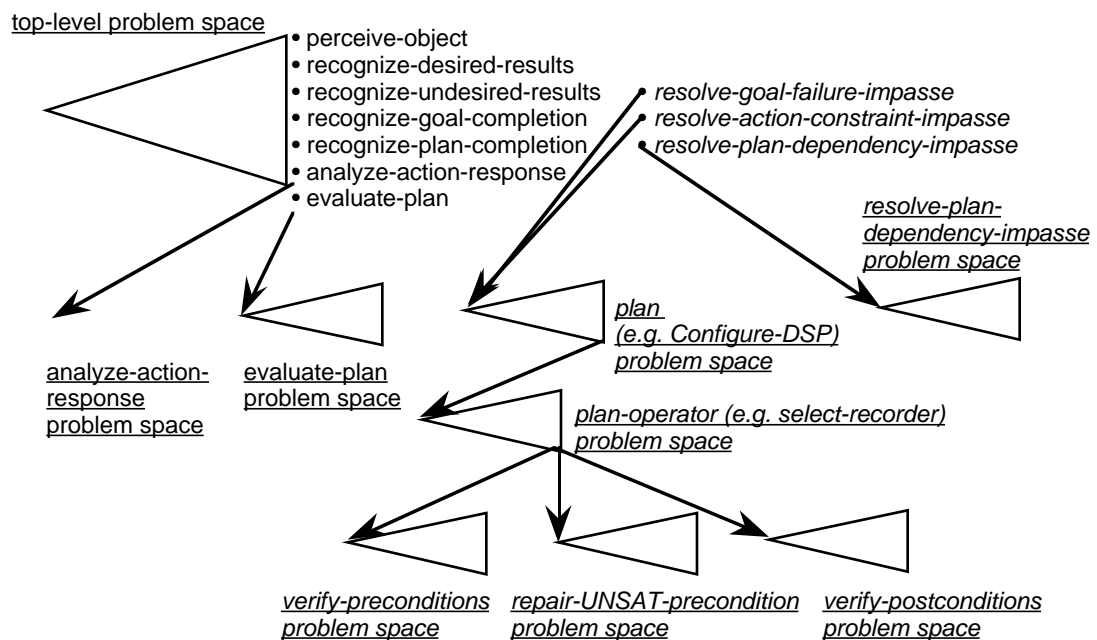
situation. An inappropriate action is classified into an impasse category, and REACT interprets the impasse by executing an expert cognitive model to explain the impasse and to suggest repairs to the procedure that will overcome it. The resulting explanation is used for tutoring the student. Plan tracking is implemented by performing the following activities:

- *Perceive objects in the environment:* REACT must continuously monitor the attributes of each of the objects in the environment. The perceive-object operator in Figure 4 initially perceives each of the objects in the environment and registers them in REACT's working memory. These attribute values are updated as the objects are observed to change state in the environment.
- *Monitor and evaluate student actions:* Each of the student's actions is observed and matched with a plan. During the plan matching, preference is given to active plans over inactive plans. Likewise, the effects of the action on the device are also evaluated to determine whether the action was successfully completed or not. If an action was unsuccessful, it is immediately classified as an action-constraint impasse. Regardless of the action's outcome, the plan containing the action is marked with the match. If the action does not match any active plan but does match with an inactive plan, then REACT recognizes that a plan-dependency impasse has occurred. All the activities for evaluating individual actions are performed by the `analyze-action-response` operator and its associated problem space; every directive-response pair is analyzed by subgoaling into the `analyze-action-response` problem space, where the plan matching and impasse recognition occurs. Subgoaling into this problem space occurs regardless of whether the student did what was expected or not, and chunks are built each time the subgoal successfully terminates. The resulting chunks eliminate the need to subgoal into this problem space the next time the same situation occurs.
- *Monitor individual goal status:* The achievement status of the individual goals of active plans is continually monitored. A plan's goals begin to be monitored when the plan becomes active; monitoring ends when the plan is inactive. The `recognize-desired-results` operator tests for satisfied goals, while the `recognize-undesired-results` operator tests for unsatisfied goals.
- *Monitor conjunctive goal status:* In addition to testing for the satisfaction of individual goals, REACT also continually monitors whether the conjunction of a plan's goals has been achieved. The `recognize-goal-completion` operator tests for the conjunctive satisfaction of a plan's goals.
- *Monitor plan execution status:* Besides monitoring whether a plan's goals have been achieved, REACT also monitors whether all of a plan's actions have been matched. The plan is marked COMPLETED once all of its actions have been observed. The `recognize-plan-completion` operator is responsible for monitoring a plan's execution status.

- *Evaluate completed plans*: If a plan marked "completed" has unsatisfied goals, then REACT recognizes that a goal-failure impasse has occurred. The `evaluate-plan` operator is used to perform this analysis--it subgoals into the `evaluate-plan` problem space where a determination is made of whether the completed plan has satisfied its goals. Chunks are built summarizing the results of the evaluation, once a determination of whether or not a goal failure impasse exists.

Impasse interpretation, which is performed by the italicized Soar problem spaces shown in Figure 4, is implemented by performing the following activities:

- *Resolve a plan dependency impasse*: REACT first determines which plans should have been active at the time that the student took the inappropriate action. The situation may have warranted a deviation from the standard procedure, e.g., the precondition of an action was unsatisfied. REACT checks the effects of the student's actions to determine whether it satisfies a precondition or else satisfies a goal of one of the plans. If either of these cases is true, the impasse is interpreted to be a situationally warranted deviation, and no further interpretation is necessary. On the other hand, if the action can not be justified by the situation, then REACT notifies the student of the violation of the plan ordering. The `resolve-plan-dependency-impasse` operator is initially selected for these tasks, and it subgoals into a problem space by the same name where the interpretation steps just described are taken.



**LEGEND:**

problem space      • operator      ↘ subgoal relationship  
 plain text -- plan tracking      *italicized text* -- impasse interpretation

**Figure 4:** REACT's problem space hierarchy

- *Resolve a goal failure impasse:* When a goal failure impasse is detected, REACT selects the plan where the goal failure impasse occurred and determines how to achieve the unsatisfied goals. It reviews the plan, which contains some history of how the student executed its actions, by checking the operators related to the unachieved goals to determine whether the student sent the correct parameters with the action. REACT internally simulates the execution of the plan by selecting the appropriate action operators, verifying their preconditions are satisfied, repairing any unsatisfied preconditions, and verifying that the actions' postconditions are satisfied. To repair an unsatisfied precondition may involve recursively selecting and applying other actions in the same manner. REACT follows this process until the plan's goals are achieved. In the process of solving the problem, REACT generates an explanation for tutoring the student about the impasse. Note that the steps just described for resolving a goal failure impasse are graphically portrayed in Figure 4 as a hierarchy of problem spaces, beginning with the operator called labeled `resolve-goal-failure-impasse`, which subgoals into the plan problem space. It is in the plan problem space that the individual plan operators are checked. If an operator is suspected as the cause for the goal failure, it is selected and subgoals are formed for the operator and subsequently to verify its preconditions and so on. As in the other cases where there is subgoaling, chunks are formed that save the results of the analysis for use in future situations where the same goal failure occurs.

- *Resolve an action constraint impasse:* REACT handles this type of impasse in much the same way as a goal failure impasse. The main difference is that it focuses on how to correctly apply a single operator within the plan rather than on the achievement of the goals of the whole plan. Hence, the `resolve-action-constraint-impasse` operator subgoals into the plan containing the suspect operator, and immediately subgoals into the plan-operator's problem space. As with the case of a goal failure impasse, subgoals are also formed to verify the operator's preconditions, repair unsatisfied preconditions, and verify its postconditions.

The problem solving in REACT bears a resemblance to the approach in CHEF (Hammond, 1990): the general strategy is to notice a failure, build an explanation for it, use the explanation to determine a repair strategy, and so on. REACT's repair strategy emphasizes treating failures related to unsatisfied preconditions, which is similar to CHEF, but REACT generates repairs on the fly rather than using a case-based approach. In addition, certain types of repairs done by CHEF are not appropriate in REACT because they would imply intervening before the impasse was clear to the student. REACT intervenes only after the impasse is clear to the student or when a potential impasse exists and allowing the student to go further would result in compounding the complexity of the impasse. If the impasse is due to violating an action

constraint, then the tutoring is immediately given since it will be clear to the student that there is a problem since the directive was rejected. In the case of plan dependency and goal failure impasses, the tutor permits the student to detect mistakes on their own up to a point, but will intervene once it is clear that the student does not recognize that there is a problem with the plan that was just executed. In any case, the philosophy behind tutorial intervention in REACT is to allow the student to experience the impasse and provide tutoring near the impasse point. So if the impasse is due to violating a plan's goal, it might seem tempting and convenient to immediately intervene at the point where the student takes the action that causes the goal to fail, but tutoring is not given until the student has completed the execution of the plan, since the goal is associated with the overall plan. This approach allows students to detect their own goal failures and correct them without intervention by the tutor. For more on why this approach to tutoring was chosen see (Hill & Johnson, 1993a; Hill, 1993.)

### **Example Revisited**

To illustrate how REACT works, we will revisit the previously used example shown in Figure 3, focusing on the action constraint impasse that occurred on lines 3 and 4, where the student issued the NRMED LD0 command that was subsequently rejected. In the plan tracking phase of the analysis, the command-response pair on lines 3 and 4 is detected by the analyze-action-response operator (Figure 4), which subgoals into the analyze-action-response problem space. Given the student action and the device's response, REACT attempts to match the command, in this case the NRMED command, with one of the active plans. As explained previously, an active plan is one that the tutor expects the student to be performing at the time that an action is observed. There may be multiple active plans at any given time, but in this case the NRMED command is matched to the active plan called Configure-DSP (Figure 2). Since the command was rejected by the simulator, the operator sets a flag indicating an action constraint impasse and the subgoal terminates. Thus, the plan tracking phase of the analysis completes, having identified that the NRMED command was expected--the Configure-DSP plan attributed to the student contains the NRMED command--but there is a clear indication that though the plan called for this step, it was not warranted by the situation.

Once the impasse has been detected, the impasse interpretation phase of the analysis begins. During the impasse interpretation, the tutor executes a cognitive model on the situation at the impasse point. Since this is an action constraint impasse, it means that one or more preconditions of the NRMED command were unsatisfied at the time of its execution. To determine the cause for the impasse the cognitive model simulates the act of choosing to send the NRMED command. In so doing it will determine which of the preconditions were unsatisfied, and once that has been accomplished it will find a way of resolving the impasse, which in this

case means finding a way of satisfying the precondition. Thus, the `resolve-action-constraint-impasse` operator is selected and a subgoal into the `Configure-DSP` plan problem space is formed. The operator corresponding to the `NRMED` command called `select-recording-device` is selected and it subgoals into the `select-recording-device` problem space (shown as `plan-operator` problem space in Figure 4.) A subgoal into the `verify-preconditions` problem space is made for each of the `select-recording-device` operator's preconditions. As it turns out, the precondition that was unsatisfied says that the recording device being selected must be in the `ONLINE` mode, but the device is currently in the `OFFLINE` mode. This is where the first part of the explanation on line 4 in Figure 3 is generated. Next, `REACT` subgoals into the `repair-UNSAT-precondition` problem space, where it is determined that issuing the `LD0 E` (enable recording device `LD0`) command will satisfy the precondition. This information is also put into the explanation on line 4 of the example. Finally, once the precondition is satisfied, the `select-recording-device` problem space simulates sending the `NRMED LD0` command (select the recording device named `LD0`), and this is also added to the explanation and this subgoal terminates. Since `REACT` has determined how to resolve the impasse, all of the subgoals in the hierarchy terminate and the impasse recognition operators resume their work with the next action-response pair.

There is one further point to be made from this example concerning the flexibility (non-rigidity) of situated plan attribution as a method for tracking the student and interpreting impasses. In lines 5 through 8, the student follows the tutor's advice and sends the `LD0` command to correct the unsatisfied precondition. Note that the `LD0` command is not part of the active plan (`Configure-DSP`) at the time that it was issued. The plan recognizer needs to be able to deal with the fact this action does not match an active plan, and the way that it does so is to check whether the directive satisfies a precondition of a step in an active plan. In so doing, the tutor verifies that the `LD0` action was warranted by the situation and does not interpret it as an impasse.

### **Tutor Improves with Experience**

`REACT` was implemented to take advantage of `Soar`'s learning capability. One consequence of having a tutor that learns is that its efficiency improves with experience. Since it is desirable to interact with the student as close to the impasse point as possible (Hill & Johnson, 1993a), a highly efficient problem solving scheme is desirable. As the tutor gains experience with different student impasses, the knowledge of how to recognize and interpret these impasses is summarized in new `Soar` productions called chunks. The chunks improve the tutor's

performance significantly, since they limit the amount of search required to solve a similar problem again.

The second consequence of having a tutor that learns is that it affects the tutor's architecture. REACT's architecture effectively changes as learning takes place: the tutor's knowledge becomes more procedural as it gains experience, and it becomes more integrated and less compartmentalized.

### Efficiency Improvement: Impasse Recognition Chunks

REACT's ability to recognize impasses improves each time it successfully recognizes a previously unobserved student action. For example, each time the REACT tutor subgoals into the `analyze-action-response` problem space (Figure 4), successfully matches an action to a plan, and determines whether there is an impasse or not, chunks are built that summarize the problem solving that occurred in that subgoal problem space. The next time that REACT sees the same action under similar circumstances, it will not create a subgoal and search for a match because there will be a set of chunks that recognize the action and know how to interpret it. The result of having chunked knowledge is that impasses can be recognized more rapidly since less search is required.

Table 1 shows the amount of time it takes to recognize an action constraint impasse involving the `NLOAD` and `SAT` commands before and after chunking. In both cases the impasse recognition time was reduced by over fifty percent using chunks. Before chunking it took approximately 0.20 seconds to recognize an impasse and after chunking to took 0.08 seconds.

Command	Recognize Impasse		Interpret Impasse		Total		Ratio: Before/After
	Before	After	Before	After	Before	After	
<code>NLOAD</code>	0.20	0.08	8.70	0.61	8.90	0.69	13::1
<code>SAT</code>	0.19	0.08	2.70	0.32	2.92	0.40	7::1

**Table 1:** Performance (in seconds) for recognizing and interpreting an action constraint violation impasse before and after chunking

### Efficiency Improvement: Impasse Interpretation Chunks

Besides using chunking to improve the impasse recognition process, REACT also takes significant advantage of chunking when interpreting the impasse. After the recognizing the impasse, the next step is to explain the nature of the impasse and determine a way to resolve it. To illustrate how chunking affects impasse interpretation, refer again to the problem space hierarchy shown in Figure 4. If an action constraint impasse is detected then the `resolve-action-constraint-impasse` operator is selected and the problem spaces below this operator are searched until an explanation is generated. In the process of generating the

explanation, chunks are created that summarize each step of the search for an explanation. For instance, in the case of the action constraint impasse involving the NRMED command on line 3 of Figure 3, each of the preconditions for the command are verified in the `verify-preconditions` problem space. In the process of verifying these preconditions, chunks are built to do this task the next time the NRMED command is involved in an action constraint impasse. Thus, whenever REACT subsequently recognizes that a student is at an impasse with the NRMED command, it will automatically fire the chunks verifying the NRMED command preconditions without needing to subgoal through the problem space hierarchy the way it did the first time.

### **Architectural Impact of Chunking**

As we mentioned at the beginning of this section, there are two consequences of having a tutor that improves with experience. The first of these consequences, improved efficiency, is the driving motivation for incorporating learning into the tutor's implementation. Improved efficiency makes it possible to conduct a timely interaction with the student. There is a second consequence of learning, however, that flows out of the use of Soar's chunking mechanism in REACT. The chunked version of REACT that emerges over time has its tutoring knowledge compiled in a form that no longer has distinguishable modules or problem spaces. The original problem space hierarchy that generated the chunks still exists but is not used unless a novel situation occurs. Instead, REACT's problem solving eventually takes place in one problem space at the top level, where it fires chunks for recognizing and interpreting student impasses.

The architectural impact of chunking, then, is that as a tutor learns, its knowledge is compiled and centralized, so to speak, into one problem space. The result is an efficient tutor that can interact in real time with a student. The resulting architecture looks different than a stereotypical intelligent tutoring system (Burns & Capps, 1988) that contains modular pieces labeled expert model, student model, tutor model, and so on. Though these are identifiable functions in REACT, they cease to have relevance from an architectural point of view once the knowledge of these functions has been compiled. Our approach contrasts with the way that Warren, Goodman & Maciorowski (1993) have proposed to engineer ITS's, whereby the traditional ITS functions are implemented in separate communicating modules. We prefer to view the intelligent tutor as an integrated agent whose architecture emerges as it learns. The functionality remains while the architecture changes.

### **Generality of Chunking-Based Plan Recognition**

There are particular features of the Deep Space Network domain, and of REACT's situated plan attribution algorithm, that are particularly conducive to chunking. These are summarized below,

in order to give the reader a better sense of how chunking-based plan recognition might be applied to other domains.

REACT's tabular representation of plans facilitates recognition of ordinary plan steps, even without chunking. The tutor does not have to search through a large space of possible student actions in order to find ones which match the student's actions. Although chunking can be employed regardless of the size of the search space, having a small search space offers practical advantages. It is not necessary to train the tutor extensively ahead of time in order to obtain reasonable real-time response, since the tutor is efficient enough to analyze unfamiliar student actions as they arise. Furthermore, chunks generated from large search spaces can often be highly specific: the left hand sides of the chunks tend to grow in size as the amount of search increases. Therefore each chunk that REACT builds is more likely to be applicable to future student actions.

In general, chunking is most effective when the problem space being searched is free of uncertainty, and the problem solver is able to determine precisely what the conditions are that lead to the result that is saved in the chunk. Otherwise the chunks that are produced may be overly general, and apply in situations in which they should not. This poses potential problems for plan recognition, which is fundamentally abductive in nature and must make plausible inferences about what the agent being watched is doing. In spite of this, we have found overly general chunks not to be a problem in the DSN domain, and expect to be able to avoid such problems in other domains as well. To justify this claim, we discuss below some potential sources of uncertainty and ambiguity, and describe why they either do not arise for situated plan attribution systems, or can be readily overcome.

If a plan recognizer were to observe the student performing an action, and jump to a conclusion that a particular plan or plan step is being carried out, the chunk that is produced will cause the plan recognizer to jump to the exact same conclusion every time, even in cases where there is evidence for an alternative interpretation. This is not a serious issue for situated plan attribution systems, because they focus on observed actions and avoid extensive reasoning about hidden mental states. Since REACT does not jump to conclusions about the student's mental state, it does not create chunks that jump to conclusions either.

There can still be a potential problem with ambiguity, though, if there is more than one known plan that can explain the student's observed actions. This could arise either because there are multiple valid ways to accomplish a given goal, or there is confusion on the part of the student as to what goal is supposed to be achieved. REACT, as well as earlier systems such as PROUST, handle this problem in part by maintaining multiple active plans, and matching them in parallel. As the student proceeds with the task, and the system is able to match subsequent student actions, the system can determine which of the candidate plans matches the student actions best.

Thus it is desirable from a plan recognition perspective to delay drawing any conclusions about the student's actions until enough observations have been made to eliminate any ambiguities.

Yet this tendency to delay interaction is precisely what the impasse-driven approach to tutoring calls for. If the student's actions do not result in an action constraint violation, then the tutor will let the student continue until he or she has completed a plan and has moved on; by this point any plan ambiguities will be eliminated. If an action constraint violation does arise, the problem can be somewhat trickier; in some domains it might be possible for the student to reach an undesired environmental state before the system has attributed a plan to the student's action. If this is the case, then it would be necessary to limit the tutor's advice to explaining in general terms how the problem arose and how to correct it, rather than to help the student proceed with carrying out a particular plan. As long as the tutor checks whether there is a unique attributed plan prior to offering advice, any chunks that are created will also check for this, and no over general chunks will be produced.

There is thus good reason to believe that situated plan attribution with chunking is robust, and can be applied to a range of different situated learning problems. As we are in the process of applying the approach to other domains, it will soon become evident whether this is in fact the case. USC / ISI is planning to develop situated tutors for operator training aboard ships. These tutors will operate within immersive virtual environments, affording a wider variety of interactions between trainees and simulated environments. JPL and USC / ISI are also examining K-12 science education as a potential application area for situated plan attribution techniques.

### **Evaluation**

A pilot study was conducted to evaluate REACT (see Hill, 1993 for more details.) We hypothesized that situated plan attribution would be able to recognize and interpret student impasses and that the resulting tutoring would help students acquire skill in the LMC domain. The study was conducted with seven students of approximately equal experience who were divided into two groups. The difference between the two groups was that Group I's students were tutored by REACT during the exercise, while the Group II students did not receive any tutoring. Each student was assigned a task to perform on the LMC simulator, where the assigned task was performed by the student six times under different starting conditions so that different types of action constraint and goal failure impasses would potentially occur.

To evaluate the effectiveness of situated plan attribution the evaluation criteria shown in Table 2 were used. The goal for this part of the evaluation was to determine how well REACT was able to understand the student's behavior in making a decision of whether or not to provide tutoring. Since REACT is an impasse-driven tutor, this means that it has to be able to detect

impasses as they are previously defined and avoids making a false detection. Hence, C1 asks whether REACT recognized all of the actions taken by the students that one would expect them to take under nominal conditions. C2 through C5 cover each of the three impasse types (i.e., action constraint violations, plan dependency violations, and goal failures). C2 and C3 address the variations of recognizing whether a plan dependency violation has occurred or not; C2 covers the impasse case and C3 covers the case where a deviant action is warranted by the situation. C4 asks whether the tutor was able to recognize and explain when the student failed to achieve the goals of a plan, and C5 asks whether individual action constraint violations were recognized.

C1: Did the tutor correctly interpret all of the "correct" actions under normal circumstances?
C2: Did the tutor correctly interpret student deviations from the default procedure when there were no situational factors requiring the deviation?
C3: Did the tutor correctly interpret student deviations from the default procedures when they were in reaction to situational factors?
C4: Did the tutor recognize when the student failed to achieve a goal? Was it able to explain the goal failure?
C5: Did the tutor recognize all action constraint violations? Was it able to explain the action constraint violation?

**Table 2:** Evaluation criteria for effectiveness of Situated Plan Attribution

## Results

The results of the pilot study suggest that situated plan attribution holds promise as a method for recognizing student impasses and for explaining how to resolve them in a satisfactory manner. During the study REACT interpreted 604 different command-response pairs (actions) performed by the students. It recognized and explicated 5 plan dependency impasses (C2), 36 actions that deviated from the plan but were warranted by the situation (C3), 17 goal failure impasses (C4), and 36 action constraint impasses (C5). In analyzing the event logs, we found that REACT did not make any misinterpretations, and it was able to make all of its analyses quickly enough for a timely interaction with the student (refer to Table 1 to get an idea of the interaction times for the action constraint impasse.) Table 3 shows the results of the situated plan attribution by subject and cumulatively.

The study also suggests that the way situated plan attribution was applied (i.e., for impasse-driven tutoring) helped students to improve their skills in the LMC domain more quickly than students without tutoring. The students from both groups reached impasses while performing the task, but there was a significant difference between the two groups in the amount of time it took to resolve these impasses. While both groups acquired the same amount of skill in cases where there was an action constraint violation, the students in Group I (tutored by REACT) resolved

impasses and acquired the new knowledge significantly faster than the students in Group II. Table 4 summarizes data from the pilot study that shows how long it took the subjects from the two groups to execute selected steps (i.e., NRMED and NLOAD) in the Configure-DSP procedure when the state of the simulator was configured to potentially cause an action constraint impasse for one or both of those steps. During the pre-test phase all of the subjects had to cope with the potential impasse for the NRMED directive without having the benefit of the tutor. Both groups appeared to fare equally poorly: two members of Group I and two members of Group II gave up trying to resolve the impasse, while the other subjects took 173, 190, and 152 seconds, respectively, to resolve the impasse and execute the directive successfully. This suggests that the two groups were equally unskilled, and it also provides an idea of how long it takes to resolve an impasse of this type.

	Cumulative Action Total	(C2) Plan Dependency Impasses	(C3) Actions Situationally Warranted	(C4) Goal Failure Impasses	(C5) Action Constraint Impasses
Subject 1	81	0	6	0	3
Subject 2	91	0	7	4	2
Subject 3	86	0	5	0	8
Subject 4	79	2	4	6	5
Subject 5	83	3	4	0	4
Subject 6	99	0	5	6	6
Subject 7	85	0	5	1	8
Totals	604	5	36	17	36

**Table 3:** Results of Situated Plan Attribution

		Pre-test M2	M3	Training Phase M4		Post-test M6
		NRMED	NLOAD	NRMED	NLOAD	NLOAD
Group I: Training Phase with Tutor	Subject 1	∞	8	11	8	2
	Subject 2	173	39	13	24	11
	Subject 3	190	29	17	11	12
	Subject 4	∞	42	26	16	76
Group II: Training Phase without Tutor	Subject 5	∞	168	49	23	10
	Subject 6	152	413	10	8	10
	Subject 7	∞	444	449	13	8

**Table 4:** Time (in seconds) to recover from potential impasses, with and without tutor

In the Training Phase, the subjects in Group I received assistance from the REACT tutor whenever an impasse was encountered, while the Group II subjects were left to struggle with the impasse on their own. In mission M3 there is a marked difference in the results between the tutored and untutored subjects. The tutored subjects who experienced the NLOAD impasse took between 8 and 42 seconds to resolve the problem, while the untutored students' times ranged from 168 to 444 seconds for resolving the problem. Though this is a pilot study with little

statistical significance, these results appear to confirm the obvious: a student who receives tutoring at an impasse point spends less time looking for a solution than one who does not have a tutor. In addition, it is interesting to note in the results in Table 4 that once a student has resolved a particular action constraint impasse, whether by the assistance of a tutor or not, the student's ability to recognize and resolve the impasse generally increases. For example, mission M4 repeats the potential impasses associated with NRMED and NLOAD, and all of the subjects exhibited improved performance. This is consistent with our hypothesis that people learn by resolving impasses (Hill & Johnson, 1993a; Hill, 1993), which is a tenet held by a number of learning researchers (e.g., VanLehn, 1988; Laird et al., 1987; Newell, 1990).

Besides evaluating how the subjects improved on action constraint impasses, an evaluation of goal failure impasses was also performed. The students in Group I were less prone to having goal failures than the students in Group II. It was observed that students who did not notice a goal failure the first time they performed a task were prone not to realize that there was one. A common but potentially serious problem that has been observed in the Deep Space Network operations domain is that the Operators may make goal-type errors that are not detected for several weeks. For instance, if an Operator does not properly perform certain calibrations on the communications equipment, the data that is acquired during a track will be adversely affected, but the poor quality of the data may not be recognized for several weeks by the scientist analyzing it. It can be extremely difficult to provide corrective feedback to an Operator when there are such significant delays between the time of the error and the time it is detected, thus it is very important to correct goal-type errors during training since the feedback in the operational environment is so delayed.

### **Conclusions**

We have introduced a plan recognition technique called situated plan attribution that we claim avoids some of the problems of other approaches, especially as applied to intelligent tutoring. Specifically, we have shown how our method is flexible enough to recognize when a situation warrants an action that is not specified by a plan. Likewise, it recognizes when an action specified by a plan is not situationally appropriate.

Situated plan attribution also addresses the issues of underconstrained and unfocused modeling in that it concentrates on recognizing students' impasse points rather than trying to generate or understand the mental states that led to a particular action. Impasse points are natural places to tutor, and the amount of processing required to recognize and explicate the impasses we have defined is reasonable.

## Acknowledgments

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Dr. Johnson was supported in part by the Advanced Research Projects Agency and the Naval Research Laboratory under contract number N00014-92-K-2015 (via a subcontract from the University of Michigan), and in part by a subcontract to the Jet Propulsion Laboratory.

## References

- Allemand, D. (1990). *Understanding Programs as Devices*. Ph.D Thesis, The Ohio State University, 1990.
- Anderson, J.R., Boyle, C.F., Corbett, A.T., and Lewis, M.W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42, 1990: 7-49.
- Azarewicz, J., Fala, G. and Fink, R. and Heighecker, C. (1986). Plan recognition for airborne tactical decision making. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 805-811, 1986.
- Brown, J.S. & VanLehn, K. (1982). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4, 1982:379-426.
- Brown, J.S., (1990). Towards a new epistemology for learning. In Frasson, C., and Gauthier, G., *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, pp. 266-282. Norwood, NJ: Ablex Publishing Corporation, 1990.
- Burns, H.L. & Capps, C.G. (1988). Foundations of intelligent tutoring systems: An introduction. *Foundations of Intelligent Tutoring Systems*. Edited by Martha C. Polson and J. Jeffrey Richardson. Hillsdale, New Jersey: Lawrence Erlbaum Associates Publishers, 1988.
- Calistri, R.J. (1990). Classifying and detecting plan-based misconceptions for robust plan recognition. Ph.D. diss., Technical Report No. CS-90-11, Department of Computer Science, Brown University, 1990.
- Clancey, W.J., (1993). Guidon-Manage revisited: a socio-technical systems approach. *Journal of Artificial Intelligence in Education*, Vol. 4, no. 1, pp. 5-34, 1993.
- Corbett, A.T., Anderson, J.R. & Patterson, E.G. (1990). Student Modeling and Tutoring Flexibility in the Lisp Intelligent Tutoring System. *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*. Ablex, 1990.
- Charniak, E. & Goldman, R. (1991). A probabilistic model of plan recognition. *Proceedings of the Ninth National Conference on Artificial Intelligence*. 1991.

- Fayyad, K. & Cooper, L. (1992). Representing operations procedures using temporal dependency networks. *Proceedings of the Second International Symposium on Ground Data Systems for Space Mission Operations*, SPACEOPS-92, Pasadena, CA, 16-20 November 1992.
- Firby, R.J. (1987). An investigation into reactive planning in complex domains. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 202-206, 1987.
- Hammond, K.J. (1990). Explaining and repairing plans that fail\*. *Artificial Intelligence*, 45, (1990) 173-228.
- Georgeff, M.P. & Lansky, A. (1987). Reactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 677-682, 1987.
- Hill, R.W.(1993). Impasse-driven tutoring for reactive skill acquisition. Ph.D. diss. University of Southern California, Los Angeles, California, 1993.
- Hill, R.W. & Johnson, W.L. (1994). Situated plan attribution for intelligent tutoring systems. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 499-505, Seattle, Washington, 1994.
- Hill, R.W. & Johnson, W.L. (1993a). Designing an intelligent tutoring system based on a reactive model of skill acquisition. *Proceedings of the World Conference on Artificial Intelligence in Education (AI-ED 93)*, pp. 273-281, Edinburgh, Scotland, 1993.
- Hill, R.W. & Johnson, W.L. (1993b). Impasse-driven tutoring for reactive skill acquisition. *Proceedings of the 1993 Conference on Intelligent Computer-Aided Training and Virtual Environment Technology (ICAT-VET-93)*, NASA/Johnson Space Center, Houston, Texas, May 5-7, 1993.
- Johnson, W.L. (1986). *Intention-based diagnosis of novice programming errors*. Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1986.
- Kaelbling, L.P. (1987). An architecture for intelligent reactive systems. *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pp. 395-410, Morgan Kaufmann Publishers, Inc., 1987.
- Kautz, H.A. & Allen, J.F. (1986). Generalized plan recognition. *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, 1986.
- Laird, J.E., Newell, A. & Rosenbloom, P.S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33, 1987:1-64.
- Maes, P. (1989). How to do the right thing. *Connection Science*, Vol. 1, No. 3, 1989.
- Munro, A., Johnson, M.C., Surmon, D.S. & Wogulis, J.L. (1993). Attribute-centered simulation authoring for instruction. *Proceedings of AI-ED 93, World Conference on Artificial Intelligence in Education*, pp. 82-89, Edinburgh, Scotland; 23-27 August 1993.

- Murray, W.R. (1986). Automatic program debugging for intelligent tutoring systems. Ph.D thesis, University of Texas at Austin, Computer Science Dept., 1986.
- Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, 1990.
- Reiser, B.J., Beekelaar, R., Tyle, A., and Merrill, D. (1991). GIL: Scaffolding learning to program with reasoning-congruent representations. *The International Conference on the Learning Sciences: Proceedings of the 1991 Conference*, pp. 382-388, Charlottesville, VA: AACE, 1991.
- Rosenbloom, P.S. & Newell, A. (1986). The chunking of goal hierarchies: a generalized model of practice. *Machine Learning*, Volume II, pp. 247- 288, edited by Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.
- Self, J.A. (1990). Bypassing the intractable problem of student modeling. In Frasson, C. and Gauthier, G., eds., *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Ablex, Norwood, NJ, pp. 107-123, 1990.
- Suchman, L.A., (1987). *Plans and situated actions: the problem of human-machine communication*. New York: Cambridge University Press, 1987.
- Tambe, M. & Rosenbloom, P.S. (1994). Event tracking in a dynamic multi-agent environment. USC / ISI tech report no. ISI-RR-393, 1994.
- Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., and Schwamb, K., 1995. Intelligent agents for interactive simulation environments, *AI Magazine*, Spring 1995, in press.
- VanLehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *The Journal of Mathematical Behavior*, 3, 1982:3-71.
- VanLehn, K. (1983). *Felicity conditions for human skill acquisition: Validating an AI-based theory*. Tech. Report CIS-21. Palo Alto, CA: Xerox Palo Alto Research Center.
- VanLehn, K. (1988). Toward a theory of impasse-driven learning. *Learning Issues for Intelligent Tutoring Systems*. Edited by Heinz Mandl and Alan Lesgold. New York: Springer-Verlag, 1988:19-41.
- Vilain, M. (1990). Getting serious about parsing plans: a grammatical analysis of plan recognition. *Proceedings of the Eighth National Conference on Artificial Intelligence, (AAAI-90)*, pp. 190-197, 29 July - 3 August 1990, AAAI Press / MIT Press, Menlo Park, CA.
- Ward, B. (1991). ET-Soar: Toward an ITS for theory-based representations. Ph.D. diss., CMU-CS-91-146, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

- Warren, K.C., Goodman, B.A. & Maciorowski, S.M. (1993). A software architecture for intelligent tutoring systems. *Proceedings of the World Conference on Artificial Intelligence in Education (AI-ED 93)*, Edinburgh, Scotland, 1993.
- Wittgenstein, W., 1953. *Philosophical Investigations*. New York: Macmillan Publishing Co., 1953.