

Intention Recognition for Task Learning

by

Phillip Michalak

Thesis Proposal

for the Degree

Doctor of Philosophy

Supervised by

James F. Allen

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

April 17, 2005

Table of Contents

| | | |
|----------|--|-----------|
| 1 | Motivation for Studying Intention Recognition | 1 |
| 1.1 | Plans and Intention | 2 |
| 1.2 | Plans and Intentions in Planning | 3 |
| 1.3 | Constrained Reasoning | 4 |
| 1.4 | Intention Recognition | 6 |
| 1.5 | Outline | 6 |
| 2 | Human Intention Recognition | 8 |
| 2.1 | Do we do it? | 8 |
| 2.2 | Types of Reasoning | 9 |
| 2.3 | Interactions | 13 |
| 2.4 | Summary | 15 |
| 3 | Procedure Learning | 17 |
| 3.1 | The Problem | 17 |
| 3.2 | Analyzing Procedure Learning | 25 |
| 3.3 | Literature Review | 26 |
| 4 | High Level Model for Procedure Learning | 39 |
| 4.1 | The model | 39 |

| | | |
|----------|------------------------------------|-----------|
| 4.2 | Trip Planning Example | 44 |
| 4.3 | Summary | 56 |
| 5 | Research Questions | 58 |
| 5.1 | Constraining Reasoning | 58 |
| 5.2 | Modeling Coherence | 61 |
| 5.3 | Inductive Learning | 65 |
| 5.4 | Indexing and Retrieval | 67 |
| 5.5 | Procedure Representation | 69 |
| 5.6 | Time line | 70 |
| 5.7 | Summary | 72 |
| | Bibliography | 73 |

1 Motivation for Studying Intention Recognition

I'd like to begin this discussion about intention recognition with some of the reasons that plans and intentions are interesting topics to study in their own right. One may claim that a key difference between humankind and other known animal species is our superior ability to form and execute plans. While other animals may display some rudimentary planning capability, the extent and depth of their ability seems to be extremely limited compared to our own. To acknowledge that other species have the same basic capability (in a limited form) begs the question of whether or not the ability to plan is one that can distinguish us from other species. Surely it is.

Humans consider themselves to be the dominant species on the planet, but it is not because of physical size, speed, or strength (which seem factors in ordering the rest of the animal hierarchy). It is our ability to reason and to form plans that gives us the upper hand. If other species were able to form and execute plans as detailed and complex as our own, the case for our dominance would be less clear. Could it be the case that our ability to form plans is merely an artifact of the architecture of the human body and mind? This is a possibility and seems to me a likelihood, but what of it? This does not make planning any less important. Achieving our present state of affairs without the ability to plan would raise a question as to its importance, but it seems extremely unlikely that this could be the case.

1.1 Plans and Intention

It seems then, that our ability to plan is an important one. What can be said of the importance of plans and intentions? Some preliminary terms are defined before this question is addressed. Intention is difficult to put into words, even though most people seem to have a clear intuitive understanding of what it is. One can speak of intentional action, in which case the action is being described as one that is performed purposefully. One can also talk about intending to act, which is a description of a state of mind. Throughout this paper I will refer to intention in this second sense, the state of mind in which one has a commitment to act. Bratman calls this the *inertia* or the *characteristic commitment* of intentions. They tend to resist casual change, making them useful for planning (described below).

Plans are slightly easier to define. They are an abstract representation of the knowledge required to achieve some goal. Some have used the term *recipe* to describe this sense of the word plan. A different use of the word is to talk of “having a plan to *A*”. In this case one has an instantiated *recipe* (the template provided by the recipe is filled with the detailed information needed to accomplish the task) in mind for achieving *A* and one has the intention to achieve *A*. It is this sense of the word plan that will be referred to throughout the paper. When it is necessary to refer to the abstract data type, the term *recipe* will be used.

The Belief-Desire-Intention (BDI) community has chosen to represent intentions as “adopted” recipes. These adopted recipes have the characteristic commitment associated with intention, and for all intents and purposes can be thought of as intentions.¹ As such, the terms *plan*, *goal*, and *intention* will be used more or less interchangeably throughout this paper. The term *plan* will be used to emphasize the entire hierarchical intention structure, and the the terms *goal* and *intention* will refer to the states of affairs that the

¹The author prefers to maintain a distinction, if only mentally, between the two. In natural language understanding domains it is generally held that discourse intentions are not equivalent to plans ([LA90]).

agent is committed to achieving.

1.2 Plans and Intentions in Planning

Coming back to our original question, what is the importance of plans and intentions to planning creatures like us? Bratman indicates that intentions play a central role in our reasoning capabilities because they are conduct controlling in a way that other attitudes are not ([Bra87]). He considers desire and intention to both be pro-attitudes, i.e. attitudes that can motivate behavior. But only intention, because of its characteristic commitment, can actually control behavior. He provides two main aspects in which intention is conduct controlling: as a *filter of admissibility* and through *means end coherence*.

A desire becomes an intention after it has been reasoned about, compared to other desires, and committed to, all against the framework of an agent's beliefs. From this point until the intention is satisfied or revoked, the intention serves as a constraint on further reasoning by screening incompatible actions. In order to act rationally, an agent should have internally consistent plans. The intentions of the agent form the *filter of admissibility* that disallows those actions that would be inconsistent. The commitment inherent in intention is the reason why such filtering is reasonable. One can desire conflicting things and still be rational, so desire is clearly not suitable for this role. Intention, however, represents those states of affairs that the agent is committed to achieving. In order to be a consistent part of the agent's plan, the action must be consistent with the agent's intentions.

Another aspect of the way that intentions control conduct is the way that they pose subproblems to be solved. The high level intentions may not initially specify the means by which they will be achieved. As time progresses, the agent must fill the plans to contain at least the level of detail that he believes to be currently required. This is the *means end coherence* constraint exercised by intention; a plan not filled in to appropriate

levels of detail will eventually become means-end incoherent.

So intentions (and synonymously plans) serve the important role of constraining future planning activities. This is a necessity because of the limited computational resources that humans have available. Our inability to reconsider every possible option at every moment often constrains us to making decisions based on prior deliberation. But they also serve another fundamental purpose; they provide a framework for social interaction.

Bratman discusses both intra-personal (among one's own plans) and interpersonal (amongst agents) coordination. He argues convincingly that neither of these would be possible without the characteristic commitment associated with intention. Knowing that agent *A* desires to be at a rendezvous point at an appointed time does not provide sufficiently convincing evidence that agent *A* will be there. As I noted earlier, an agent can desire contradictory things, so there is no guarantee that agent *A* will do anything to achieve his desire to be at the rendezvous point; he may have some other conflicting desire that he acts upon. The agent may in fact do nothing at all because that is deemed a better option. In contrast, Agent *A*'s intention to be at the rendezvous point *does* allow us to expect that Agent *A* will attempt to be at the rendezvous point (since intention is conduct controlling). Furthermore, one can also expect that Agent *A* will be in a position to act successfully on that intention because of the *means-end coherence* pressure provided by intention. In effect, these two primary roles of intention both support the expectation that Agent *A*'s intention to do something will result in *A*'s doing that something.

1.3 Constrained Reasoning

Implicit in Bratman's discussion of intention is their role in constraining reasoning so as to make rational agency achievable. Because we are resource bounded creatures, we require an organizing principle for our cogitation, and intentionality fits the bill nicely. A number of researches have adopted this view of agent behavior and have developed a class of systems known as BDI agents that place primary importance on beliefs, desires,

and intentions. Let us consider the ways that intentions constrain reasoning.

One way that intentions facilitate rational agency in the face of bounded resources is by limiting the types of information that we consider. Intentions are characterized by commitment, and will necessarily limit the information that we pay attention to. We do not have infinite processing power, and must therefore limit our information intake using some decision criteria. Were we to haphazardly choose which items to consider and which to ignore, we would risk ignoring information about those goals to which we are committed. This could, in turn, result in suboptimal completion or the outright failure of said goals. It is clear, that if we are committed to achieving these goals, that we must be more principled in our consideration of information. We must limit our information intake in such a manner that we have sufficient resources to consider information relating to our intentions. This is not to say that the only things we consider are those that are directly related to our intentions. We may consider many topics unrelated to our intentions during the course of a day, but never so many and for such periods of time as to negate our ability to achieve those goals to which we are committed (at least in the case of “rational” behavior).

Another way that intentions facilitate rational behavior is by placing bounds on the amount of reasoning that we perform. Our limited resources must be allocated judiciously in order to guarantee that the goals to which we are committed can be achieved. Our commitment to these goals requires that we perform those activities which will achieve them, which further requires that at some point we “stop thinking and start doing”. Were we to reason for arbitrary periods of time without regard for the constraints imposed by our commitments, we would sometimes fail to achieve our goals because the time had passed when they could be achieved. It is clear that our intentions play a major role in achieving rational agency. Subsequent portions of this proposal will consider mechanisms by which an agent’s commitments could constrain reasoning in the manner discussed above.

1.4 Intention Recognition

This section motivates intention recognition as a topic of research. From a practical perspective, we want software to interact with users more intelligently. The extent to which this can be achieved by merely organizing the content presented to the user is limited. At some point software systems must begin to augment the user's efficacy by providing appropriate resources and services at the correct time. More generally, software systems have more information available to them than the keystrokes, mouse-clicks, and commands that are explicitly represented. All of these are situated in patterns of behavior that suggest intention. This is a high level information source that has been largely ignored in simple user interfaces. As interfaces become more complex, as they begin to incorporate modalities of human communication, intention recognition becomes a key to understanding what the user is really doing in order that the system may collaborate.

A second independent motivation for plan recognition stems from the sentiment that any theory of intelligent agency should incorporate the ability to interact with other agents. Principled interaction requires coordination, and coordination requires that agents act in predictable ways. As discussed above, intentions provide this requisite predictability, but in order to capitalize on it, agents must first be able to recognize it. Only then can coordination efforts be successfully initiated based on mutual expectation.

1.5 Outline

This chapter has provided some basic motivation for research in intention and plan recognition. Humans are planning creatures, and computer systems that interact with them should take advantage of that fact. Intentions play a central role in framing the development of future plans, and as such will be good predictors of future behavior. Recognition of these will allow systems to interact more naturally and with more focus.

Chapter 2 discusses the types of reasoning that humans use to perform intention recog-

dition, and suggests that the traditional boundaries placed on reasoning processes are perhaps not so clear as they have been made out to be. Chapter 3 proposes task learning as a suitable domain for further research on the interaction between intention recognition and other reasoning processes. Chapter 4 proposes a high level model for the procedure learning task that incorporates natural language input. Chapter 5 describes the research questions posed by this model, mentions some of the relevant literature, and proposes a time table for addressing these issues in a task learning framework.

2 Human Intention Recognition

This chapter discusses human intention recognition and the types of reasoning required. We have access to a number of general reasoning faculties which we combine in novel ways to accomplish intention recognition. Among these are abductive, deductive, and meta-level reasoning. Before discussing their roles in the intention recognition process, I suggest that humans do actually perform intention recognition and provide some reasons why this is likely the case.

2.1 Do we do it?

The opening chapter contains an argument that intentions are useful for resource bounded rational agents. Humans are clearly resource bounded and in general act rationally. But do humans make use of intentions? The experiments of Schmidt, Sridharan and Goodson ([SSG78]) provide supporting evidence that humans do. Their (human) subjects were given written descriptions of a hypothetical person's actions. Their subjects formed plans to explain the actions of the hypothetical actors. Furthermore, the subjects were observed to have poor recall of those actions that did not fit well into the hypothesized plans, indicating that the subjects used these plans to encode the behavior of the actor. I think it hard to argue that humans are not intentional creatures.

The process of intention recognition yields as it results the likely intentions of another agent. As Bratman has argued, these are useful to the perceiver because they are conduct controlling. This endows the recognizer with the ability to predict the behavior of the

observed agent. Furthermore, it allows the agent to make its own plans based on those predictions. This type of interaction is fundamental to social coordination. It is almost certain that humans perform intention recognition so that they can make predictions about the behavior of other agents with whom they interact. Even a completely anti-social agent needs to reason about the likely actions of other agents in order to avoid potential interaction.

2.2 Types of Reasoning

This section considers the types of reasoning that play a role in intention recognition. The following generic notation for describing inference rules is to be interpreted as meaning that ϕ can be inferred whenever ψ_1, \dots, ψ_n can be inferred.

$$\frac{\phi}{\psi_1, \dots, \psi_n}$$

2.2.1 Abductive Reasoning

This form of reasoning allows one to infer causes of a particular observation. Formally, abduction is defined by the following inference rule.

$$\frac{\phi(A)}{\phi(x) \rightarrow \psi(x), \psi(A)}$$

Observing that $\psi(A)$ holds may lead one to believe that $\phi(A)$ holds since ϕ is a cause of ψ . This is not a sound inference, since there is no guarantee that $\phi(A)$ is true whenever $\{\phi(x) \rightarrow \psi(x), \psi(A)\}$ are true. However, this type of reasoning is often reasonable for proposing explanations of observed data.

For instance, if one observes in the morning that there is no hot water available with which to shower, one might infer that the pilot light on the hot water tank has gone out. This provides a reasonable explanation of the observation, since having the pilot light of the tank out necessarily implies that hot water will be unavailable at some point. It may be the case, however, that this inference is not warranted. The hot water tank may

have an electronic ignition without a pilot light, which cannot therefore have gone out. In any case, there are multiple potential causes of the observation, and abduction allows one to reason backward from the observation to the possible causes.

In the specific case of intention recognition, one is reasoning not about cause and effect, but about motivating behavior and resulting action. One reasons about potential intentions that would motivate the observed behavior. Viewed thus, abductive reasoning lies at the heart of intention recognition. Abduction provides a direct link from the observed actions to possible reasons for that behavior. While other reasoning processes play a role in intention recognition, abduction plays a crucial role because of its observation driven nature. One could conceivably reason strictly deductively from intentions to expected actions and eventually find an expected action that matched the observation, but this is an extremely inefficient way to achieve the effect of abduction. In both cases, a hypothetical “what if” is being asked in the presumption of an intention. Abduction makes sure to ask “what if” about only those intentions that could explain the observation (according to the knowledge base).

Intention recognition has been used in dialogue understanding ([AP80], [Car90], [CG93]), which has itself been cast as an abductive problem ([HSAM93]). Hobbs and Stickel represent the semantic content of a sentence as a conjunctive clause in first order logic. Interpretation is taken to be the derivation of this clause from the current knowledge base, using abduction as required to “explain” literals that cannot otherwise be derived. Such assumed literals become the new information that is provided by the sentence. Mayfield characterizes intention recognition as a sequence of abductive inferences (with the help of deduction).

2.2.2 Deductive Reasoning

Deductive reasoning is a sound form of inference that allows us to reason from causes to effects. Formally, deduction is defined by the following inference rule.

$$\frac{\psi(A)}{\phi(x) \rightarrow \psi(x), \phi(A)}$$

This type of inference plays a major role in our every day reasoning, including our intention recognition activities.

As an example, consider a dialogue domain where a computer user corresponds with technical staff members in order to retrieve a file that was accidentally deleted. In response to a staff member's question "When do you need the file restored?", the user might respond "The project is due on Friday".¹ In order to properly interpret the user's response, the staff member would need to use abduction to conclude that the file is related to the project, and deduction to conclude that the file should be restored before the due date since it cannot be submitted before it is restored.

In general, deductive reasoning will facilitate abductive reasoning by deriving new facts that allow existing abductive chains to be extended or to tie them together. In the previous example, the answer to the question cannot be answered by abductive reasoning alone. Deductive reasoning is required for the staff member to translate the project constraints into constraints on when the file must be restored.

2.2.3 Meta-Level Reasoning

Meta-level reasoning is the use of other reasoning methods in contemplation of the reasoning process. A clear example motivated by the previous chapter is the use of reasoning processes to determine when to stop reasoning. It is often the case that there are so many inferences that could be made given a set of observations and an initial knowledge base that they cannot all be made in a reasonable amount of time. Humans are generally able

¹This example borrowed from [May92]

to stop reasoning when it is appropriate to do so.

There are a number of ways that meta-reasoning can affect human intention recognition. How detailed should the inferred intentions be? How many intentions should be inferred? When is there too little information to infer reasonable intentions? These types of questions are ones that we implicitly and occasionally explicitly address during reasoning.

2.2.4 Causal Reasoning

Deductive reasoning allows one to reason from cause to effect, and abductive reasoning allows inference in the opposite direction. Both inference types are a component of causal reasoning, which is used to reason from evidence to its likely causes and effects. Causal reasoning is more general than either of these types of reasoning, and as in abductive inference, there is no guarantee that the results of causal reasoning will be correct. Though far from infallible, this method of reasoning is frequently employed and does play a role in intention recognition beyond its incarnation as abduction and deduction. As a simple example, consider observing an officemate don headphones to listen to music. One could infer that the officemate wishes to avoid being engaged in casual conversation, and has consequently chosen to deter conversation by listening to music. This is clearly not a deductive inference, since the officemate listens to music through headphones in many different situations. It is also not an abductive inference since wishing to avoid casual conversation does not imply that the officemate will use the headphones; a trip to the library, a “quiet please” sign, or a number of other measures would accomplish the same goal.

2.2.5 Planning

Planning is employed in intention recognition in order to generate predictions about what someone might do next. Once an observer has attributed a set of (potential) intentions to the observed, expectations can be generated about future behavior by way of planning from the perspective of the observed. By adopting the (attributed) intentions

of the observed, the observer can plan to achieve the (attributed) intentions in the same manner that the observed would. The actions generated by this 'vicarious' planning process will serve as expectations about the future behavior of the observed. The degree to which these expectations are fulfilled by the observed subject will provide feedback to the observer about the accuracy of the postulated intentions (or the assumptions made in the vicarious planning episode).

As an example, consider the train station employee working at an information booth.² A reasonable intention for the harried commuter who asks if the 10:00 train to Toronto has left yet is that they wish to catch the train. The employee may reason that pursuant to this intention, the commuter may purchase a ticket, proceed to the loading area, and embark on the train. This vicarious planning episode may prompt the employee to respond with the location of the train departure (especially if this information has changed) in addition to whether or not the train has departed. This same behavior would be reasonable from any person in the train station, not just the employee who has answered these questions so often that the responses are second nature. The ability to plan from the perspective of another person (i.e. with their intentions and beliefs as a backdrop) allows humans to achieve these sorts of helpful, predictive behavior.

2.3 Interactions

The discussions above show the role of various reasoning capabilities in intention recognition. A natural question to ask is how tightly integrated these forms of reasoning are. I believe that the interactions between them are complex to the point of suggesting that they are actually aspects of a single reasoning capability. I offer only some anecdotal evidence that this may be the case.

Consider the following scenario taken from a television show.³ A husband and wife have

²Taken from [AP80]

³Episode #317 of *CSI: Crash and Burn*

been brought to the emergency room and treated for smoke inhalation. The wife dies and the husband is slow to recover; the cause in both cases is eventually determined to have been carbon monoxide poisoning. A special investigator is somehow made privy to the details of the situation and the house is examined for clues as to how this couples' carbon monoxide levels might have become so high. The investigator examines the house and finds a broken damper in the bedroom of the afflicted couple. Eureka! The broken damper leaves the fireplace unable to vent the fumes generated by the fire, which therefore fill the room instead, thereby causing the carbon monoxide poisonings. This is causal and deductive reasoning at their very best. In addition to the broken damper, a black substance in the back of the fireplace is determined to be activated charcoal, which it seems, will produce carbon monoxide upon burning without the warning smell of smoke.

At this point, the investigator begins to reason that activated charcoal does not place itself in the fireplace, so someone must have placed it there. What could this agent's motivation have been? This is exactly the question that intention recognition asks for each action attributed to an agent. The investigator thus begins an intention recognition process to determine what possible intentions the hypothetical agent could have had. The prospects are grim. The agent is either a moron or a calculating killer, with the situation suggesting the latter. In order to reach this conclusion, the inspector reasons that the agent may have placed the charcoal in the fireplace (knowing that there was a broken damper) so that carbon monoxide fumes would be generated without the warning smell of smoke. These fumes would fill the room instead of venting outside as a consequence of the broken damper, and would eventually cause the carbon monoxide poisonings. The agent may therefore have intended that the room's occupants succumb to carbon monoxide poisoning. This intention recognition sequence makes use of the very same causal and deductive reasoning sequence that the inspector used earlier, outside the context of intention recognition.

This example suggests to me that causal reasoning, for instance, is one of a number of

general reasoning capabilities that we have available for stand-alone use (e.g. outside the context of intention recognition in this example) or in service of other forms of reasoning (e.g. during the course of intention recognition). Assuming for now that these forms of reasoning are actually distinct processes, is this causal reasoning capability really available for use by these processes or is it instead *duplicated* within them? The latter option seems unlikely for architectural reasons. Furthermore, strange behaviors are possible if there are any differences in the multiple instances of the causal reasoning faculty. Consider a situation in which the intention recognition version of causal reasoning is unable to make a particular inference that the stand-alone version of causal reasoning could make. Then an agent reasoning in an intention recognition process might be unable to draw a causal link between two events, thereby ignoring certain avenues of thought. If however, the agent were to stop performing intention recognition and to start performing 'real' causal reasoning then that particular inference could be made. The agent could perform another mental context switch back to intention recognition and continue from the impasse. While this is a possible model of reasoning, it is an unappealing one.

This example suggests even more though. Causal reasoning is used in service of intention recognition, but intention recognition is also used in service of causal reasoning. At the point in the story-line where the inspector reasons that an agent must have placed the charcoal into the fireplace, the investigator calls upon intention recognition to reason about why such an action might have been taken. The intention recognition process, in turn, recursively invokes the causal reasoning faculty. This example suggests to me that extremely intricate patterns of interaction are possible, and that these faculties may not in reality be distinct, but rather aspects or byproducts of a single general reasoning capability.

2.4 Summary

Models of human intention recognition are not the focus of this proposal, so I will not attempt to answer the questions about human reasoning raised in this chapter. These do,

however, provide motivation for investigating the interactions between various reasoning processes in machine implementations. Since humans appear to be relatively successful at recognizing intentions and reasoning in general, models of human reasoning are relevant as sources of inspiration for machine reasoning.

In the next chapter I discuss the problem of task learning, which relies on many different reasoning processes in order to learn a task procedure from human instruction. This problem provides a rich framework for examining the interaction of these different processes, and it is an interesting problem in its own right. Intention recognition has not yet been explicitly incorporated into treatments of task learning, and I propose that it provides a situation of mutual benefit. Intention recognition offers improved learning capability to task learners, and task learning provides an interesting combination of reasoning processes and an opportunity to study their interactions.

3 Procedure Learning

This chapter describes the procedure learning problem and proposes it as a good framework for studying intention recognition. It incorporates many different forms of reasoning and provides an opportunity to examine their interactions. Intention recognition will provide some important benefits for knowledge-based procedure learning systems.

3.1 The Problem

A system that performs procedure learning is one that observes the performance of a particular task and learns enough about the task to satisfy some criteria for successful learning. Different systems place different restrictions on the form of observations and the form of the knowledge learned about the task. These differences tend to stem from the different tasks that the systems are designed to handle and from the different techniques used to achieve the learning. In most cases, the criteria for learning is that the system be able to execute the learned procedure in order to accomplish the task. This usually entails learning the appropriate procedure steps and can additionally include learning their constraints, preconditions, and effects.

In the context of this proposal, the tasks will be executed by a human user interacting with a computer. The user will provide input to the system in the form of user interface actions and spoken dialogue. The user will use dialogue to explain what actions or goals are currently being pursued and how those goals fit into the context of the larger task.

The architecture that I outline in Chapter 4 will make use of any background knowledge about the task domain and the user description of the task to learn a task model and a procedure for achieving the task. This task model will incorporate the various steps that the user performed and any causal relationships between them that it can infer. Furthermore, it will learn a hierarchical task model by grouping these actions into appropriate sub-tasks whenever possible. The research questions posed by this model are discussed in Chapter 5.

3.1.1 An Example

The following simple example should serve to make the types of expected user interaction more explicit. Consider a person who frequently travels for business, and wishes to train a helper to find the details of flights and accommodations in order to accommodate trip planning. Such a user (U) might engage in a dialogue similar to the one in Figure 3.1 in order to teach an agent (A) an appropriate procedure for finding the necessary information.

Examining the dialogue of Figure 3.1 provides some insight regarding the types of reasoning that are typically involved in this type of procedure learning. This dialogue ¹ is typical of the types of interaction that I expect a procedure learning system to operate on. In the following discussion, A will be taken to be a computer program.

The first thing to note is that the user (U) makes some of the temporal and causal relationships between subtasks explicit during explanation of the procedure. In this example, U provides an initial high level structure for the task: first find tickets, then find accommodations, then find area attractions. This structure provides an initial constraint on the order of the subtasks. Subsequent dialogue may provide additional causal constraints, e.g. 'Now that we've identified hotels we can find nearby attractions'. The hotel

¹The term dialogue is used loosely here, since A brings little to the conversation. In general, A will provide better feedback about the learning process. The purpose of this example is to show the expected mode of user interaction.

U: I'm going to show you how to plan a trip.
A: OK.
U: First you find tickets, then you find accommodations, then you find area attractions during the time of the visit.
U: Let's find tickets first.
U: First you go to Travelocity.
<User navigates to Travelocity in web browser>
U: Then you select 'Search Flights' from this menu.
<User selects a menu entry>
U: Then you type ROCHESTER in the FROM field.
<User types ROCHESTER in text box>
U: Then you type ATLANTA in the TO field.
<User types ATLANTA in text box>
U: Then you specify the departure date.
<User types a date into the departure field>
U: And select a preferred departure time.
<User selects a time preference from time menu>.
U: Do the same for the return date.
<User types a date into the return field>
<User selects a time preference from the time menu>.
U: Then you press the 'Search Flights' button.
<User presses button>
U: Now you gather flight information.
U: For each flight, find the airline and flight number, the departure date and time, and the arrival date and time.
U: Let's look at the information from one of the flights>.
<User selects a link for a flight>
U: The airline and flight number is here>
<User selects the text for the airline and flight number>
U: The departure date is here.
<User selects the text for the departure date>
U: The departure time is here.
<User selects the text for the departure time>
...

Figure 3.1: A dialogue snippet for teaching a procedure

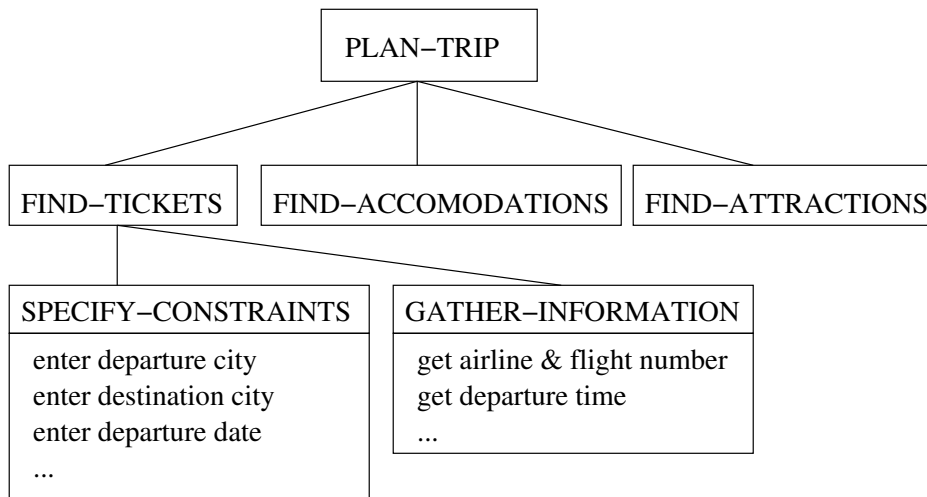


Figure 3.2: Task Structure for Trip Planning Procedure

finding must occur before the attraction finding because the user wishes the attractions to be nearby the hotel.

In addition to making explicit the structure of the task, U also provides clues as to which portion of the task model is being described. The user indicates that the find-tickets sub-task is going to be described by saying 'Let's find tickets first'. Presumably there will be a point in the dialogue where the user says something like 'Now let's find accommodations' when the instruction pertains to finding accommodations.

These types of helpful behavior are characteristic of good teaching. Merely providing the sequence of actions may enable a learner to repeat the sequence of actions under exactly the same conditions, but it hampers the learner's ability to modify or troubleshoot the procedure should something go wrong. In general, this explicit description of task structure and causality is a resource to be utilized in robustly learning a procedure. Causal reasoning therefore plays a role in procedure learning.

Not all causal constraints are apt to be made explicit, however, and it is because of this that intention recognition becomes important in procedure learning. The learner

should understand as best it can how each action that the user takes serves to achieve the task goal and how it interacts with other actions. Intention recognition seeks to find the intentions/plans of the user that caused the user to perform the observed actions (including speech). Knowledge of these intentions/plans gives the procedure learner a deeper understanding of the task components and the role that their performance plays in achieving the overall task. As an example, consider the user's instruction to proceed to Travelocity after indicating that finding tickets is the current sub-task. A system that infers the intention of specifying preferred travel times via the date/time entry actions may learn preferences (inductively) from repeated invocations of the learned procedure. Such a system could also use learned preferences as reasonable defaults in cases that the user did not specify the particular time of day for the flight.

In addition to helping understand the causal relationships between task components, intention recognition can also help to determine the overall task structure. When U is observed to say 'Now you gather flight information', A is faced with the problem of placing this sub-task into the context of the current task model. As long as A can infer that the intention for gathering information about the flights is so that one can examine and compare flights for the purpose of selecting or finding a good one, then it make sense to place this subtask as a child of the 'find-flights' sub-task. Often there are multiple cues for the appropriate task structure. In this case, both discourse coherence and intention recognition suggest that 'gather-information' is a sub-task of 'find-flights'. Additionally, intention recognition can suggest the generation of intermediate sub-tasks. If A recognizes that the departure/destination date/time/city information are entered with the intention of specifying trip constraints, then a sub-task can be hypothesized that comprises these individually described sub-tasks. This would yield a task model similar to that of Figure 3.1.1.

Another type of reasoning that I've alluded to but haven't explicitly mentioned is the planning that learner A uses in order to help solve the problems of recognizing causal relationships and task structure. Each time that the user specifies a sub-task to perform,

the learner may be reminded of certain facts that are related to the performance of that sub-task. In these cases, the learner can plan 'vicariously' to see what types of behavior are expected of a user attempting to achieve that task. This planning sets up expectations about future behavior that observations should confirm. There is a strong interplay here between the expectations generated by planning and the intention recognition mentioned above. Intention recognition hypothesizes intentions that explain the observations, while the vicarious planning places constraints on the types of intentions that we expect to see. When intention recognition grounds itself into these expectations, a learner has determined the causal relationships between various parts of the sub-task.

3.1.2 Intention Recognition in Procedure Learning

There are some informal arguments in the example above regarding the benefits of intention recognition in procedure learning. This section provides in more detail two arguments for the incorporation of intention recognition into procedure learning. Intention recognition provides additional information to the task learner without burdensome annotation, and procedure learning provides opportunities for the study of intention recognition.

One obvious reason for incorporating intention recognition into the procedure learning task as it is described above is the presence of natural language dialogue. This natural mode of communication with other agents requires intention recognition processes to provide the intended meaning of each utterance. The user will have different goals throughout the course of a training session that may generate similar surface form realizations. A user could say "let's work on finding tickets" to introduce a new sub-task as part of a trip planning task. This same utterance could, however, also indicate that the user wishes to return to a previously discussed sub-task, possibly created as part of an initial high-level description.

Intention recognition provides a mechanism for ignoring spurious actions in the training

sequence. Sometimes the user will perform actions that have no bearing on the task at hand. If a system can recognize these actions as part of typical user behavior that is unrelated to the current task (e.g. checking email or moving a window to view the system clock), then it can be filtered from the task. This particular use requires a high degree of confidence in the recognized intention and in its unrelatedness to the task. The automated experimentation technique described in [AJRS02] could be used in this situation to determine whether or not the suspect action truly is spurious. In principle, however, good intention recognition can provide this capability. A more general (and more desirable) solution to this problem is to recognize that these “spurious” actions are actually part of other (possibly interleaved) plans. Both approaches would improve the quality of the system interface, as they would allow the user to speak and act more naturally (e.g. “I’m going to teach you how to buy a book. First I’m going to get a cup of coffee.”). The second solution, however, has much greater potential for natural interaction since humans frequently pursue multiple plans simultaneously.

Perhaps the most important benefit that intention recognition brings to the procedure learning task is the ability to determine implicit causal relationships. As described above, there will often be cases when the user leaves implicit the causal relationship between actions in the task. Determining these relationships improves the system’s understanding of the task. Understanding the *reasons* that certain actions or sub-tasks were performed allows the system to reason about alternative methods of achieving those goals. For example, merely recording that the user double-clicked a particular icon may suffice so long as the icon is available, but in a different environment where the icon is no longer present the causal intent motivating the action (i.e. launching an application) is necessary in order to complete that portion of the task. Additionally, the better understanding afforded by knowledge of the causal constraints allows the system to communicate more effectively with the user about the task. If the user were to instruct the system to delete a step that achieved the precondition for a second step, the system could indicate this dependence and ask how the precondition should be achieved in the absence of the deleted step.

The relationship between procedure learning and intention recognition is a symbiotic one. On the one hand, intention recognition provides to the procedure learning task the benefits described above. On the other hand, procedure learning provides a good framework for studying intention recognition.

There are multiple places in the procedure learning task that intention recognition can be applied, and therefore there are multiple places where it can be studied. Intention recognition is used to interpret the speech acts provided as input by the user in two different ways. The first is to determine the intent of the user in speaking the utterance (i.e. what action does the user want the system to take?). The second is to determine how the desired action relates to the current task model. Intention recognition is also used to determine how the user interface actions relate to the task model. These different types of interaction provide different vantage points for studying intention recognition.

There are also a number of different reasoning processes that are involved in the procedure learning task. Causal reasoning is used in intention recognition as well as in other processes. Indexing and retrieval processes are vital for remembering past procedures and adapting them to new tasks. Expectations are generated by a *vicarious planning* process as new sub-tasks are identified, and are matched by the intentions produced by intention recognition. Expectation generation and intention recognition should intuitively perform their tasks more effectively if they work together than if they work completely independently. This is true also of inductive reasoning, the type of reasoning that allows one to generalize from specific instances to the more general case. The procedure learning task provides an environment where the same task can be taught with different parameter bindings or with the same parameter bindings but by many different users. A user could teach the system to buy a book, clothing, or art supplies. Many different users could teach the system their own procedure for shopping for each of these items. These different training sessions provide an opportunity for inductive learning and for its interaction with intention recognition.

In summary, procedure learning and intention recognition provide mutual benefit. On the one hand, intention recognition can improve the usefulness and performance of procedure learning systems. On the other hand, procedure learning provides a good framework for studying intention recognition and its interaction with multiple different reasoning processes.

3.2 Analyzing Procedure Learning

This section discusses some criteria for evaluating existing procedure learning systems. Since this is a proposal for incorporating intention recognition into procedure learning, the degree to which intention recognition is already present in each system will be discussed. The burden on human designers will be assessed by examining the background knowledge, annotation, and user input requirements for each system. The expressiveness of the resulting learned procedures will be assessed as well as the actual knowledge learned about the tasks.

The *background knowledge* required by a system is that knowledge about the task or the task domain which must be present in some explicit representation in order for the system to function. Some systems require virtually no background knowledge and are able to learn procedures inductively from example interactions alone. Other systems require very specific information about the task domain and sometimes about the particular task itself in order to perform properly. Those systems that require large amounts of information will obviously be harder to port to new domains, especially those with significantly different task structures. There is often a tradeoff between the amount of background knowledge required and the specificity of the knowledge learned or the applicability of the learned procedure. In the preceding example, the background knowledge comprises all of the facts that the system would make use of during intention recognition, planning, etc.

The *annotation* requirements of a procedure learning system can vary from none to a very detailed specification of the tasks being learned. For the purposes of this analysis

the annotation requirements of a system are those pieces of the system input that are not generated solely through the interaction of the user and system in order to accomplish the task. In the example described above, the verbal communication provided throughout the training session would be considered a form of annotation, albeit a very natural and unobtrusive one, because it is not strictly necessary in order to accomplish the task at hand.

The *expressiveness* of the resulting learned procedures is important for determining the types of domains in which the task learner can function. The task model representation determines in large part what types of procedures can be learned; the learning algorithms play a complementary role.

Finally, the type of the knowledge that is learned will also be considered. In some cases, the knowledge is logical and is easily reasoned about, while in other cases it is strictly probabilistic, and potentially harder to analyze, reason about, or debug. Some systems come away with a thorough understanding of the procedure components, while others know relatively little about what is happening in the task. The latter type of system is typically best suited to domains where reaction is more important than deliberation.

3.3 Literature Review

The following discussion of implemented procedure learning systems ² is based on the criteria outlined above. The systems are designed for different domains, and hence have different requirements for the types of input, annotation, and resulting procedures. They are described here to provide a representative sample of the methods that have been applied to procedure learning.

²The systems described here were suggested in an unpublished literature review by Hyuckchul Jung

3.3.1 Lau et al.

Sheepdog ([LBCO04]) is a system that learns technical support procedures by observing multiple “experts” perform the same procedure under multiple operating conditions. The system is probabilistic in nature and requires very little background knowledge.

The system defines a *trace* of an expert interacting with the computer to be a sequence of context/action pairs $t_i = \{S_i, A_i\}$. Each pair represents the context of the computer system (the windows visible on the display, knowledge of the preceding actions, and “other contextual information”) at each time step and the action that the user took as a result of examining that context. A *procedure* is defined to be a graph of procedure steps in service of a single goal. Each procedure step represents a particular point in the procedure, e.g. “opened the TCP/IP properties dialog, setting the DNS server”. An expert following a procedure examines the current system context (including the result of previous actions) and decides which procedure step to go to next. An appropriate action is selected which is calculated to advance the expert to the desired step. However, actions can fail or have unpredictable consequences depending on the context of the system (e.g. navigation to a web page may fail because there is no network connection).

The system learns a procedure by *aligning* the traces of multiple experts onto a procedure. An alignment is a function that maps each element of a trace to a procedure step. Multiple elements of the trace can be mapped to the same procedure step, indicating that the expert attempted the same goal multiple times during the trace. The formal statement of the problem is to learn a procedure model Ψ given traces T_1, \dots, T_n such that each trace is optimally aligned onto the procedure. The alignments are learned using Input-Output Hidden Markov Models (IOHMMs), which include context information in both their transition probabilities and their output probabilities. The probability of transitioning to state i at time t from state j at time $t-1$ is given by $P(X_t = i | X_{t-1} = j, S_t)$, where S_t is the state information at time t . Similarly, the probability of output observations (actions) are given by $P(A_t | X_t, S_t)$; they are conditioned on the current HMM node

(the procedure step) and the current context. The alignment process is a variation on the traditional EM algorithm for Hidden Markov models in order to include the context information.

This system makes use of no intention recognition reasoning in the traditional sense. Statistical information is accumulated which allows the system to determine the most likely procedure step that the user is following given observation of their action. The reasons that an action might have been performed are not considered.

The nicest feature of this system is its limited background knowledge and annotation requirements. The only information that the system needs outside of observation traces is the number of nodes in the IOHMM. Determining this information, of course, is a bit of an art, and the authors provide no mention of how they chose the number of nodes for the models in their experiments. There is no annotation requirement. The system is dealing directly with the user interface events generated by the system (and parsing them into semantic actions, e.g. moving a window, double-clicking an icon, etc.).

The generality of the tasks learned is somewhat impaired by the lack of domain knowledge. Specifically, tasks whose higher level context exerts significant influence on sub-tasks may not be amenable to this type of learning if that higher level context is not propagated. The characterization of the context (including “other contextual information”) is imprecise, and does not specify how context is propagated to subsequent actions. Maintaining an entire history of actions and screen entities would make for unreliable probability estimates (sparse data) and would be space inefficient for long sequences. The authors assume that the appropriate context will be available on the current screen, but this is not likely to be the case in general, and does not conveniently lend itself to tasks that require parameterization. Since the task parameter is generally unlikely to be displayed on the screen, it will not be a part of the context, and hence cannot play a role in determining the actions that the learned procedure chooses.

The form of the knowledge that is learned is difficult to reason about. There is no obvious way to adjust a learned procedure that consistently errs in a particular situation. Manually tracking down the combinations of probabilities and context that generate the error is difficult, as is adjusting the parameters to correct one specific aspect of behavior.

3.3.2 Lent and Laird

KnoMic (Knowledge Mimic, [vLL01]) is a system for learning air combat behaviors from observation. KnoMic is a specific instantiation of a general architecture for learning by observation. The architecture allows various implementations for each of its components: the environmental interface, the environment (simulation), an observation generation process, a condition learning process, an operator classification process, and a knowledge translation process.

An expert issues commands to the environmental interface, which are executed in the environment, and the state of the environment is passed back to the expert through the environmental interface. The observation generation process observes the expert interaction with the environmental interface and records the commands issued to the environment and the information returned to the expert. The resulting observation traces are annotated (either during or after the interaction) by the expert, yielding sequences of actions that correspond to the operators in effect at each time step. The condition learning process requires the annotated trace as input and learns the preconditions and goal conditions of each operator. Together with the observation trace, this operator information is provided to the operator classification process, which determines the *repeatability* of the operator (i.e. under what conditions, if any, the operator can be repeated). The resulting operators are then passed through the knowledge translation process to produce an executable format that can be used to interact with the environment in place of the expert.

The operator is the organizational unit of the KnoMic task structure, and is a derivative of the original STRIPS operator ([FN71]). KnoMic operators contain preconditions that indicate when they may be applied, and goal conditions that indicate when they have achieved the desired effects. These goal conditions are necessary because unlike STRIPS operators, KnoMic operators do not directly manipulate the environment with an effect list. Instead, they issue commands which are evaluated by the environment and which may or may not have the desired effect. Operators are hierarchical; the children of an operator can represent alternatives or a sequence for successfully accomplishing the operator goal. The framework places the restriction that only one operator at each level of the hierarchy may be active at any moment. In addition, an operator may only be activated if its parent operator is active. These constraints appear to force the resulting executable knowledge to be applied in a predictable manner; i.e. corresponding to paths through the operator hierarchy.

The ModSAF battlefield simulator ([CSC⁺93]) provides the environment for KnoMic. The observation generation component consists of a user interface component that allows the expert to mark operator changes as they occur. KnoMic uses the Find-S specific-to-general induction learning algorithm ([Mit97]) for learning the preconditions and goal conditions of the operators. Finally, the learned operator information is translated into Soar production rules in the knowledge translation phase of KnoMic, and the resulting productions are used by the Soar engine to interact with the ModSAF simulator to test the learned operators.

On the surface KnoMic processing seeks the answer to a question similar to one that intention recognition asks: why was an operator applied? KnoMic, however, seeks a much more literal answer than does intention recognition: the operator's goal conditions and preconditions rather than motivating goals or intentions. Beyond this surface similarity, there is no intention recognition occurring in the KnoMic system. There is no role for it, since the entire task structure is specified by the expert during trace annotation.

KnoMic requires as background knowledge the operator hierarchy for which goal conditions and preconditions should be learned. Though learning by observation reduces the burden of knowledge engineering, it has not entirely erased it; the requirement that the operator hierarchy be laid out is non-trivial, especially given the constraints placed on the hierarchy by the KnoMic assumptions (no two operators from the same level of a hierarchy may be simultaneously active; an operator may be active only if its parent is active). The burden of specifying operators has, however, been reduced considerably by eliminating the need to manually specify goal conditions and preconditions. These are the elements that the system learns inductively through multiple annotated traces.

Each trace must be manually annotated to indicate each transition between operators. In a complex domain like air combat, this may not be possible at execution time, so a trace annotation tool can be used instead of the execution time interface for annotation. This type of annotation is standard for state of the art procedure learning methods. In one way shape or form, most procedure learning techniques require information about the transitions between tasks. Sheepdog (described above) is a notable exception, though its “tasks” are primitive actions and therefore require no segmentation.

KnoMic is capable of learning conjunctive and disjoint disjunctive condition sets using the Find-S specific-to-general algorithm, but is unable to learn overlapping disjunctive conditions. Domains in which the latter frequently occur will not be feasible to this particular learning system (though a different learning algorithm could fix this). Loops and conditionals of a limited fashion should be possible, if unwieldy. If one operator plays the role of the loop test, and a sub-operator plays the role of the loop increment (and is repeatable) then this pair should function as a loop. Conditional branches can be modeled by a single node with a sub-operator for each branch. These structures, however, have to be manually placed in the operator hierarchy, while the loop conditions and branch conditions (both operator preconditions) are learned automatically. In general, the procedures that are learned by this method seem limited to reactive behaviors. The authors state that “Tasks involving extensive internal planning or time delays between

trigger and action are difficult since these aspects of the task aren't apparent to the observation system". It appears that the domains to which this system is applied must have carefully structured operator hierarchies in order for all of these assumptions to hold. The air combat domain, for example, generally involves a good deal of reasoning (about friendly and hostile contacts, uncertainty, aircraft capabilities, etc.), but by making certain assumptions much of the reasoning component can be removed. If one assumes that the aircraft will be tasked by a separate dispatching agent, then the operators need only recognize the condition that an intercept request has been received instead of reasoning about when it would be appropriate to perform an intercept. The operators that implement the intercept are largely reactive to the state of the aircraft and the environment.

3.3.3 Garland et al.

Garland et al. ([GRR01]) have implemented a system that allows knowledge engineers to specify hierarchical task models by example. These task models are declarative knowledge that can be used to determine when a sequence of actions accomplishes a task or to generate such sequences. These task models are useful for various different types of reasoning; the authors cite the discourse interpretation, plan recognition, and action selection algorithms of Collagen ([RS98], [RSL01]) as typical examples of situations where such knowledge is necessary. They argue in the same vein as van Lent and Laird that "learning by demonstration" will reduce the burden of knowledge engineering.

This system provides an environment for life-cycle management of task models. Task models are learned inductively from multiple examples and then subjectively evaluated. Such evaluation might motivate further changes to the examples and/or manual editing of the task model. Changes to the model (automated or manual) may have unforeseen consequences, so the system will pro-actively notify the user when early examples are no longer consistent with the model as a result of recent changes. The inductive task model learning component is most relevant to this proposal; its characteristics are described

below.

The task model is essentially that of Grosz and Sidner ([GS86]), comprising actions and recipes. Actions can be primitive (directly executable) or non-primitive (abstract), requiring decomposition. Actions have types, and each action type has an associated set of parameters. Actions do not, however, have preconditions or effects. A recipe decomposes a non-primitive action into a set of steps that will achieve it, and several different recipes may achieve the same non-primitive action. Recipes also have a set of constraints which impose partial temporal orderings on their steps, as well as equality relationships among step parameters.

The learning algorithm takes as input an annotated sequence of actions, and produces a task model. The input can contain non-primitive action place holders (representing some undisclosed sequence of events for accomplishing it), but is primarily composed of primitive actions. The sequence of actions is segmented to indicate its hierarchical structure (the non-primitive actions and the actions taken to achieve them). Additionally, three mappings *unequal*, *optional*, and *unordered* are provided as input. These indicate respectively whether pairs of action parameters are constrained to be equal, whether an action is required, and whether pairs of actions in the same segment must be ordered. The authors experiment with providing different combinations of these three mappings in order to determine where knowledge engineering effort is most useful. They conclude from their experiments that providing equality relationships between parameter values alone significantly reduces the number of examples required to learn the correct task model, while the others provide only a small additional improvement. The authors note that the parameter value equalities are typically easy for humans to recognize and encode. An example of this type of annotation is to mark equality between the α parameters in the following *PastaRecipe* recipe decomposition of *PreparePasta*(α_1): $PastaRecipe \rightarrow Boil(Water : \beta_1), Cook(Water : \beta_2, Pasta : \alpha_2)$.

Intention recognition is not incorporated into this system, though the authors indicate that the resulting task models are suitable for use by intention recognition processes. Because the action knowledge does not include preconditions or effects, it is difficult to use the resulting task models to reason about novel user plans. As a result, a plan recognition system based on this type of task model will require a complete task hierarchy that describes every way that each task can be accomplished. Such assumptions are made by a number of plan recognition systems (e.g. [Bau94], [Kau90]).

The annotation complexity of this system is higher than in the previously described systems. The user of this system will normally provide task traces that are annotated with equality relations between action parameters in addition to the hierarchical action structure. Though equality constraints are not required, they significantly reduce the number of examples needed to train the task model so it is generally in one's best interest to include them. The hierarchical structure of each example must also be spelled out explicitly through annotation. This requirement implicitly uses knowledge of the task structure. The authors consider this knowledge to be more readily available for examples, and the system is based "on the conjecture that it is often easier for people to generate and discuss *examples* of how to accomplish tasks than it is to deal with task model abstractions." The resulting circularity (learn a task model based on knowledge of the example structure) is evidenced by the iterative development process that they've espoused. Examples provide the basic structure for the task model, and subjective measure of the resulting task model motivates modification/creation of examples to extend it.

The task models learned by this system are somewhat limited in their usefulness for actual task execution. Since the action descriptions specify no preconditions or effects, it is difficult or impossible to tell when an action has had its desired effect solely by observing the environment. Furthermore, there is no native support for conditional branching or loops, which are useful procedure elements many practical domains.

3.3.4 Angros et al.

Angros et al. describe a system (Diligent) that learns tasks for the purpose of training other human users the same task. They leverage direct specification, programming by demonstration, and autonomous experimentation to reduce the knowledge bottleneck associated with learning task models. Of the systems described, the philosophy of this research is most similar to that of this proposal: causal knowledge is important to task understanding and need not be stated explicitly. The method by which it is obtained, however, is ultimately different than the intention recognition process that this proposal suggests.

A task comprises a set of end goals, a set of steps, ordering constraints on the steps, and causal links between the steps. The end goals represent the conditions that must hold for the task to be considered complete. The steps may be primitive actions or sub-tasks, and are partially ordered by the ordering constraints. Each causal link represents the production by one step of a particular precondition for another task step (or for task termination). The causal links provide valuable information for explaining the role of a step in completing the task, a feature that is useful for teaching the task.

In addition to tasks, Diligent reasons about operators, which model the preconditions and effects of the domain actions that it observes. The operator provides a more complete description of action effects and preconditions than the task model, which contains only those that are observed to have been in causal relationships with other task actions.

Diligent learns by observing an expert's interaction with a simulator. The simulator is assumed to accept commands from a user interface or from Diligent, and is expected to report each user action by providing the simulation state, the action taken, and the effects of the action. This information is used to learn an initial model which is then available for manual update.

Each action observation within the sequence is represented by an operator ³ that models the preconditions and effects of that particular action instance. A version space algorithm ([Mit82]) is used to learn the preconditions and effects of each operator during an autonomous experimentation phase that occurs after the observation phase. The experimentation phase consists of multiple trial interactions with the simulator. The action/operator sequence is repeated once for each of the actions in the sequence (each time starting with the same initial state that was observed for the original observation sequence). Each trial sequence has a different step removed. This provides multiple examples of the state before a particular action/operator application and the effects afterward. Ideally this process will generate some positive instances of each operator (whereby the action effects are “similar” to the original observation) and some negative instances, providing additional information about the preconditions and effects of each operator.

Diligent next augments the original demonstration with the precondition information acquired through experimentation. Using this precondition information, (some of) the causal relationships can be determined. From the initial state, the sequence of actions is analyzed to determine which steps produce the preconditions of later steps. This same analysis also generates a partial order on the actions. Once the task model has been generated, a user can manually edit the task model to correct for missing/extraneous causal links.

The authors claim that an extension for learning hierarchical task models is possible, but do not elaborate sufficiently to convince the reader that the mechanism is viable. In any case, this work introduces an element that is missing or only implicitly present in the other systems: the explicit determination (by the system) of the causal relationships between parts of the task. van Lent and Laird ([vLL01]) do address operator goal con-

³It is unclear whether each action type is represented by an operator or if each action instance is represented by an operator. The former would seem to offer more information about the preconditions and effects of particular action, but the latter may be more practical for learning when the action parameters can alter the effects of an action.

ditions and effects, but don't provide an explicit mechanism for exploiting this knowledge.

Diligent does not make use of intention recognition, though it does address one of the same problems through a different mechanism. Autonomous experimentation identifies causal relationships between task steps in a manner that is suited to Diligent's target domains; little background knowledge is required in order to draw these connections.

This system requires no background knowledge or annotation, largely because it does not provide automated support for hierarchical task learning (at least as described in [AJRS02]). The only input required from the expert are the actual actions that one would take to perform the task. Because this system learns operator characteristics solely from environment observations, it is not suited for domains where complex reasoning guides action. Diligent is unable to determine relationships that are not readily apparent from examining the environment state.

3.3.5 Summary

These systems have slightly different motivations for learning a task model: for execution, for teaching, for informing other reasoning processes. None of these incorporate intention recognition in any significant sense, though some of the problems that they address could benefit from it.

Causal relationships between task steps are addressed explicitly only in [AJRS02]. Diligent, they argue, must understand the task well enough to be able to instruct others, and hence must understand the role of each task step. While teaching may not be the goal of all task learning systems, learning a robust model of the tasks should be. Intention recognition provides a mechanism for learning relationships similar to those that Diligent does. While intention recognition does not entirely eliminate the requirement for structural annotation, it does it does provide a much more natural way to achieve it. Dialogue is a natural mode of communication with a high degree of structure. The

degree to which this structure can be leveraged to infer the hierarchical structure of a task model remains to be seen.

The next chapter discusses a high level model that incorporates intention recognition into a task learning framework. This model alleviates the burden of manually annotating examples for task structure and learns causal relationships between the actions in the procedure.

4 High Level Model for Procedure Learning

This chapter proposes a high level model for learning procedures from a combination of user interface interaction and natural language instruction. An example interaction is described which illustrates each step of the model and the various reasoning processes that occur.

4.1 The model

The following model of procedure learning incorporates dialogue as a key component. This system assumes the presence of some amount of domain knowledge. The degree to which this knowledge is present affects the degree to which implicit relationships between sub-tasks are learned and the accuracy of the hierarchical task model. Both of these issues could be addressed in subsequent “practice sessions” during which the system attempts to perform the task that it has learned in the presence of the human user. This aspect, however, is not explicitly addressed in this high level model, though it is expected to be part of a complete procedure learning model.

The procedure learning model is represented here as a hierarchical goal structure that is activated any time that a procedure should be learned. This goal structure is depicted in Figure 4.1. While this goal structure could be implemented as some number of interacting procedures, I prefer to think of it as a hierarchy of goals, some of which will remain active through multiple activations of the top level *know-procedure* goal. This

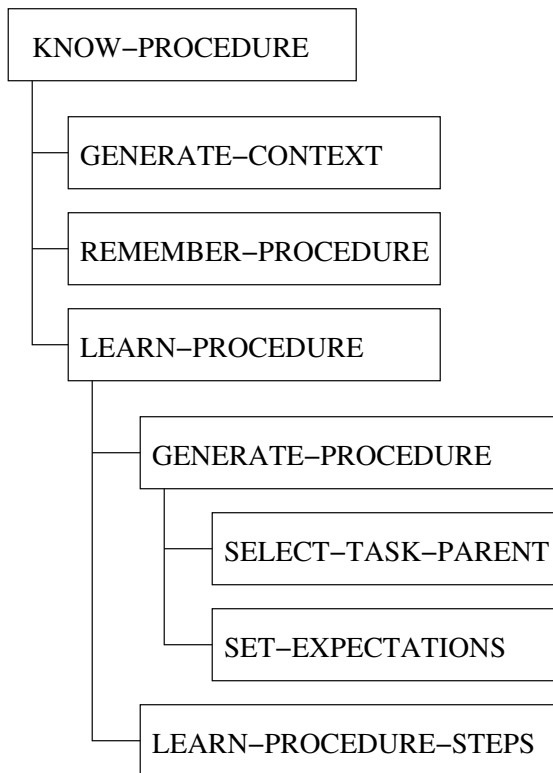


Figure 4.1: The goal structure of the high level model

formulation at least lends itself to implementation in an agent architecture. Each of these procedure learning goals will be described below.

The result of the procedure learning goals is a task model for accomplishing the task demonstrated by the user. The task model is hierarchical, with each task in the model being characterized by a goal and a procedure for achieving the goal. The procedure can be a task decomposition or a primitive action sequence. The conceptual task structure is depicted in Figure 4.1. Each task also has associated with it a sequence of expectations. These expectations are the goals which the system expects to be adopted by the user in performing the task.

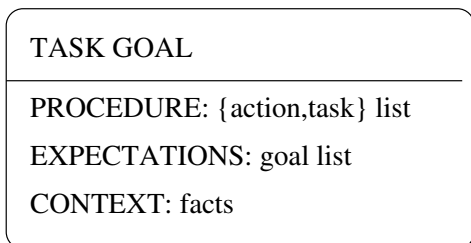


Figure 4.2: Task structure

1. know-procedure ?x satisfaction-cond(?x, ...)

This is the top level goal that is generated each time that the system infers that the user wishes to teach a procedure. The actual description of the task that the procedure accomplishes is informally modeled here as the sequence of logical predicates *satisfaction-cond*. Whatever representation is selected should support subsequent retrieval of the procedure. The procedure learning sub-goal *remember-procedure* is responsible for recalling the procedure based on descriptions similar to the one described by *satisfaction-cond*. As an example, a book buying procedure might be represented as the entity ?x which satisfies:

$$\forall b \exists e \text{ book}(b) \wedge \text{buy_event}(e) \wedge \text{purchase_object}(e, b) \rightarrow \text{achieves}(?x, e)$$

2. generate-context ?x satisfaction-cond(?x, ...)

This goal is part of the strategy for constraining the reasoning processes in procedure learning. The reasoning that can be performed in support of the task is limited by the context associated with that task. The context should include the facts that are related to the task (as determined by the background knowledge available to the procedure learner). It is conceivable that an empty context will be generated for entirely novel tasks. Throughout the rest of this proposal, context will be treated as if it provides a “sandbox” in which reasoning should operate. This is perhaps not the best analogy for a context mechanism, and is certainly not the only way that such a “context” mechanism might function. One could think of the context mechanism as a process of identifying and making more salient those inferences and facts that are related to the task, perhaps using a spreading activation mechanism. Under this interpretation, the context items are those which

would be “activated” by such a process. Switching focus from one sub-task to another might then involve a “context switch”, in which the context items associated with the new task are activated. Such activated facts and inferences are preferred to less recently activated items. This view of context requires an additional mechanism for bounding the length of inference paths. A primitive model is suggested in Chapter 5.

3. remember ?x satisfaction-cond(?x, ...)

This goal deals with the remembering of learned procedures that (almost) satisfy the *satisfaction-cond*. This goal is always generated as a sub-goal of the *know-procedure* learning goal and serves two distinct functions. The first is to remember procedures that can serve to provide expectations for learning a new procedure. The remembered procedure will ideally be similar enough to the new procedure that its structure can be used to frame expectations. The second function is to remember a sub-task of the current task model that remains unelaborated. These unelaborated tasks will most often be created during a high-level description of the overall task and elaborated upon at a later time. An example of each use will be provided below.

4. learn-procedure ?x satisfaction-cond(?x, ...) ?remembered

The function of this goal is to actually learn a new procedure. It’s not always the case that having a *know-procedure* goal will give rise to a *learn-procedure* goal, specifically in case an exact match for the *remember* goal is found and the procedure does not need to be learned. If the procedure does need to be learned then this goal is posted. Most of the procedure learning action occurs as a result of the subgoals generated by this goal.

5. generate-procedure ?x satisfaction-cond(?x, ...) ?remembered

This goal causes a new task place-holder to be constructed and to be committed to memory (so that future *remember* goals will be able to access it). The *generate-procedure* sub-goals will initialize the task and place it in the current task model.

6. set-procedure-expectations ?x satisfaction-cond(?x, ...) ?remembered

This goal causes expectations to be specified for the completion of the task (possibly including completion conditions). These expectations play key roles in structuring the task model (through interactions with processes from *select-task-parent*). These expectations may be generated from a remembered procedure or from reasoning from first principles. The planning process will answer the question 'How would I accomplish this task?' and hence 'What do I need to know in order to accomplish this task?' These suggest suitable starting points for expectations. The context created by *generate-context* can be used to constrain the reasoning processes used to generate these expectations. Note that since the task model is hierarchical and the task has been placed into the existing structure, there is a hierarchy of context information available as well.

7. *select-task-parent* ?x satisfaction-cond(?x, ...) ?remembered

This goal finds the appropriate place in the current task model to place the new task place-holder. There are multiple sources of information available to guide this operation. The user provides clues in the form of the utterances (e.g. words like "first", "next", etc.), the expectations from existing tasks in the task model may match the goal of task being placed, and intention recognition processes may indicate the appropriate point of attachment. The combination of abductive intention recognition and expectation generation is reminiscent of early work in intention recognition by Allen ([AP80], [All83]) and more recently by the plan reasoning of Ferguson ([Fer95]).

8. *learn-procedure-steps* ?x satisfaction-cond(?x, ...)

This goal causes procedure steps (either sub-tasks or primitive actions) to be recorded. As procedure steps are assigned to a task, they are matched against its expectations. Knowing that expectations have been met provides important information for structuring the task model and assigning future actions to tasks. It is to be expected that multiple different instances of this goal will be active simultaneously, particularly in top-down learning sessions where multiple sub-tasks are mentioned as part of a high level overview of a larger task.

The following example will show in detail how this model will learn a procedure (task model) for performing a simple hierarchical task. The details of implementation for each part of the model are ignored in the following discussion, and the ideal behavior is described. The following chapter will discuss various research questions whose answers will provide the basis for implementing the model components.

4.2 Trip Planning Example

In order to observe how the model learns a procedure, recall the trip planning example from the previous chapter (shown again in Figure 4.3). This example provides a simple (yet hierarchical) structure useful for illustrating the interactions of the various parts of the learning framework. Additionally, some of the types of reasoning involved in learning this model have been discussed from the vantage point of a human reasoner. This treatment of the example takes one step toward a machine implementation, using the earlier discussion as background and motivation.

The user begins by informing the system that a procedure for planning a trip is about to be taught. The system is a dialogue agent, and this initial interaction with the user is parsed and interpreted by the same processes that will handle all of the user's spoken input. Among these interpretation processes will be an intention recognition component that determines the user intention for each utterance. This first utterance from the user is interpreted as the manifestation of the user's intention to teach a procedure for trip planning. Recognition of this intention and subsequent deliberation by the system will likely result in cooperation, i.e. the adoption of an appropriate learning goal. The system begins its learning process with the posting of the resulting *know-procedure* goal.

Upon adopting the goal of knowing a procedure for *planning a trip*, the system will follow the high level procedure described above for achieving this goal. Note that the first problem that we encounter (aside from interpreting the utterance) is to generate indexing information for this procedure so that it can be remembered in future situations.

U1: I'm going to show you how to plan a trip.
A2: OK.
U3: First you find tickets, then you find accommodations, then you find area attractions during the time of the visit.
U4: Let's find tickets first.
U5: First you go to Travelocity.
<User navigates to Travelocity in web browser>
U6: Then you select 'Search Flights' from this menu.
<User selects a menu entry>
U7: Then you type ROCHESTER in the FROM field.
<User types ROCHESTER in text box>
U8: Then you type ATLANTA in the TO field.
<User types ATLANTA in text box>
U9: Then you specify the departure date.
<User types a date into the departure field>
U10: And select a preferred departure time.
<User selects a time preference from time menu>.
U11: Do the same for the return date.
<User types a date into the return field>
<User selects a time preference from the time menu>.
U12: Then you press the 'Search Flights' button.
<User presses button>
U13: Now you gather flight information.
U14: For each flight, find the airline and flight number, the departure date and time, and the arrival date and time.
U15: Let's look at the information from one of the flights>.
<User selects a link for a flight>
U16: The airline and flight number is here>
<User selects the text for the airline and flight number>
U17: The departure date is here.
<User selects the text for the departure date>
U18: The departure time is here.
<User selects the text for the departure time>
...

Figure 4.3: The trip teaching interaction revisited

This *indexing problem* suggests itself as one of the research questions for this proposal, and will be discussed in Chapter 5.

The system will adopt the further goal of *generate-context* as part of the process for achieving *know-procedure*. Generating the context for the task of trip planning is part of a mechanism for directing the reasoning that is performed in service of learning a procedure for the task. These context items are preferred to non-context items during inference. This method is patterned after the frequently observed psychological phenomenon of *priming* ([MS71], [CL75]), whereby recently accessed facts (and closely associated ones) are more quickly accessible than they would normally be. The context plays an additional role in *constraining reasoning*, by providing a measure of *interestingness*. I discuss this further in Chapter 5. For the purposes of this discussion, assume that the generated context includes facts about trips (that they involve travel and hence transportation, that they are durative events, have an origin and possibly multiple destinations, etc.) and facts about planning for trips (i.e. planning for transportation, planning for durations spent at destinations, etc.). The context generated for a particular task will be a function of the knowledge that is present in the knowledge base, the associations present between facts in the knowledge base, and the particular algorithm for selecting appropriate contextual information.

Once the context has been generated for the trip-planning task, the system will adopt the goal of remembering an existing procedure for trip planning (*remember-procedure*). Pursuing this goal raises again the two questions described previously. The context generated prior to remembering attempts should serve as a guide for constraining reasoning. The results of remembering may be an exact match (the system believes that it already knows how to plan a trip, possibly from an earlier training episode), a partial match (the system believes that it knows how to plan for events similar to trips), or failure. In the first case, the system may short-circuit its attempt to learn a new procedure after appropriate dialogue with the user, but it may also proceed with learning, albeit with more information about what to expect during training. A partial match may also pro-

vide additional information. For this example, assume that no procedure is remembered, and that further learning is required.

The system will adopt a *learn-procedure* goal if it decides to continue with the learning (perhaps using a remembered procedure as context). The adoption of this goal drives the learning process, using the context generated by the previous goals to constrain and guide the learning process. The *generate-procedure* and *learn-procedure-steps* goals are adopted in order to achieve the *learn-procedure* goal.

The *generate-procedure* goal is achieved by updating the current task model with a new task representation for the new task (trip-planning). This includes creating the task representation, associating with it the indexing and context information generated previously, associating with it an appropriate set of expectations, and inserting the task into the model. At this point in the processing, the task is created and is indexed accordingly in memory. The context generated by *generate-context* is associated with this task to constrain future reasoning involving the task. Subgoals are adopted to insert the task into the model and to generate the appropriate set of expectations.

The *set-procedure-expectations* goal causes the system to reason about an appropriate set of expectations for a trip-planning task. If an existing procedure had been remembered, its sub-tasks can be used at this point as potential expectations. The system should reason about their potential for accomplishing the current task before accepting them as expectations. The system can also reason from background knowledge about the ways that it would accomplish the task. The result should be some set of goals that the system expects to see the user pursue in order to accomplish the task of trip-planning. Assume for the current example that the system reasons from scratch that a trip involves traveling, and hence forms an expectation that the user will arrange transportation. Assume that the system has no further information about traveling scripts, and therefore neglects to set an expectation that the user will arrange accommodations for the duration of the

stay.

Now that the context and expectations have been generated, the system adopts the goal of selecting the appropriate insertion point in the task model (*select-task-parent*). In the current example, there is no existing task structure in the model, so the task for trip-planning is inserted into the model with no parent. This task now has the current model focus.

Now that task is situated in the task model, the goal of learning its component procedure steps is adopted (*learn-procedure-steps*). This goal is persistent across future invocations of the learning procedure, terminating once the task expectations have been met by observed actions or the addition of sub-tasks.

At this point, the processing generated by the user's uttering of U1 is complete. This processing may occur in parallel with the dialogue interpretation processes, but task placement must occur before user interface actions can be processed or subsequent tasks can be placed. Subsequent utterances will be treated less pedantically, with detail proportioned according to the relevance of the various sub-goals to the particular utterance.

The system may respond at any point during the training session, though a response will likely not be generated for every user utterance. At this point, however, it is important for the user to know that the system has adopted the goal of learning the trip-planning task, and a content generation capability should be called upon for this purpose. During the rest of this example, no system responses will be generated, though in actual practice the system should adhere to principles of cooperative conversation, e.g. grounding. ([GS86], [RLR⁺02]) If the system is particularly confused about some aspect of the task (e.g. the appropriate parent for a new task, the appropriate task recipient of an observed action, the relevance of a proposed sub-task, etc.), an appropriate query may be generated. If the uncertainty is deemed not to have serious ramifications for the

learning, the system may make a default decision with the expectation of correcting mistakes during a practice session. No such queries will be generated in this simple example.

At this point the users second utterance U3 can be processed. The user's intention of indicating the sequence of tasks to accomplish the trip-planning is recognized, and the system adopts the goals of knowing procedures (*know-procedure*) for each of the tasks mentioned: finding tickets, finding accommodations, and finding area attractions. The intention recognition process may recognize that the tickets to be found are likely transportation tickets based on the top-level task and the expectations surrounding it. If so, the find-tickets task can contain this more specific understanding of the ticket finding sub-task. The remainder of the dialogue is devoted to finding tickets, and so shall the majority of the following discussion.

The same procedure described for processing the trip-planning task is applied to the ticket-finding task, but there are some important differences, so the relevant steps will be revisited. The *know-procedure* goal for ticket-finding causes the *generate-context* goal to be adopted. The context for ticket-finding would include facts about tickets (e.g. tickets have prices, travel tickets have origin and destination locations, etc.) and ticket-finding facts (e.g. likely locations for obtaining travel tickets, etc.). Whenever possible, the specific facts pertaining to the top level task's domain (i.e. travel) should be included, though general facts are also important, especially in the absence of the more specific facts.

Now that the context has been generated, the system attempts to remember a procedure for finding (travel) tickets. It's possible that a previous learning session generated a procedure for purchasing plane tickets through Expedia.com. In this case, the *remember-procedure* may recall this procedure, which it can then use to frame expectations during the *set-expectations* goal processing.

The *learn-procedure* and the *generate-procedure* goals are subsequently adopted. A new task representation for the ticket-finding task is created, and the *set-procedure-expectations* goal is adopted. Without the learned Expedia procedure, the system might reason about the steps necessary to find travel tickets in order to generate expectations. Such a process might result in expectations that a ticket supplier must be visited, and that a ticket specification must be provided to the supplier. With the availability of the learned Expedia procedure, these expectations will likely already be present as sub-tasks: 1. navigate to Expedia, 2. input ticket specifications. In either case, the system should generate expectations similar to these from either general planning capabilities or from prior learning.

Given that the top level goal is trip-planning and that the only task currently in the task model is the trip-planning task, it is evident that the adoption of the *select-task-parent* goal should cause the trip-planning task to be selected as the parent of the new ticket-finding task. There is something more to this process that is worth mentioning, however. The expectations of the trip-planning task included arranging transportation, and the goal of the ticket-finding task plays a key role in such an arranging. An intention recognition process that makes this connection will be able to generate an appropriate causal link between the ticket-finding and the transportation arranging. Furthermore, the transportation arranging expectation is (at least partially) met by the ticket-finding sub-task. Keeping track of expectations that have been met allows the system to determine task structure, and can also provide clues that the system is misunderstanding when expectations are frequently violated. The ticket-finding task is inserted into the model as a child of the trip-planning task, and any discovered causal links are included. The ticket-finding task now has the focus in the task model.

Similar processes are performed for the accommodation-finding and attraction-finding tasks. When it comes time to select the task parents for the accommodation-finding task, the ticket-finding task will be considered first as it holds the focus. The utterance U3 contains information on the structure of the trip-planning task. The three

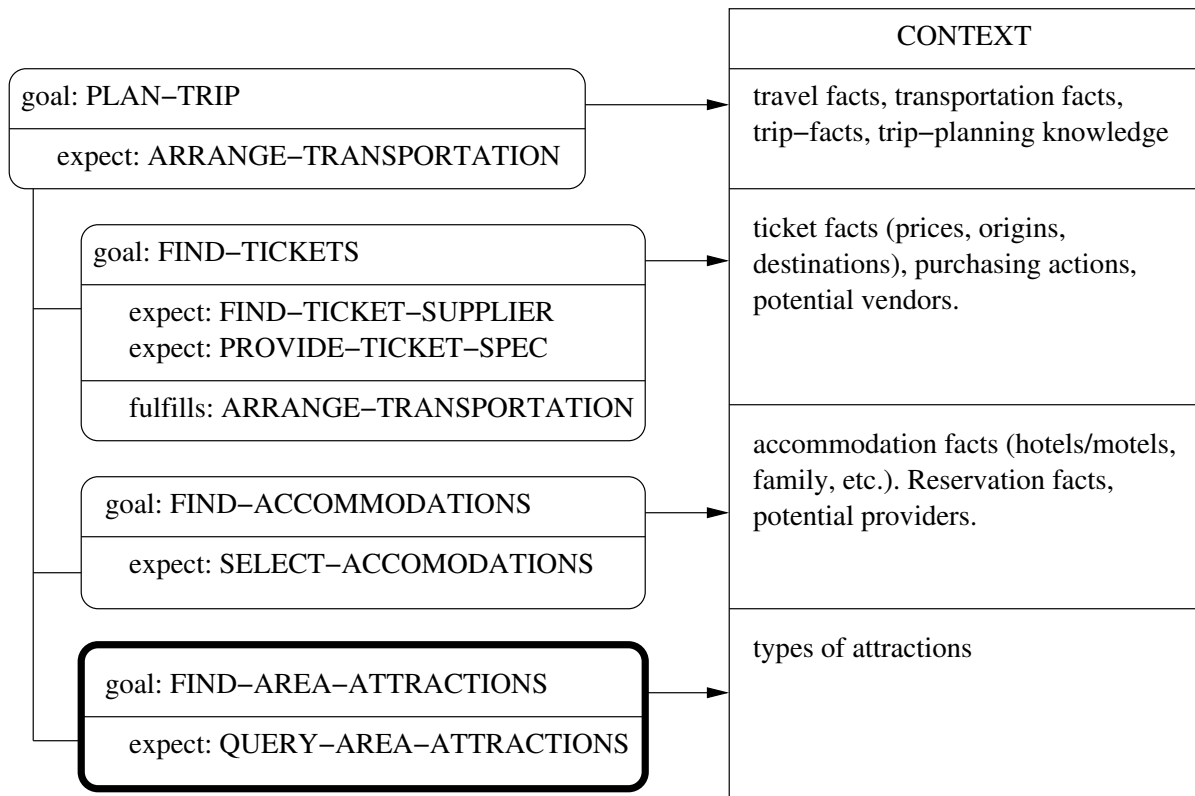


Figure 4.4: The task model after processing up to utterance U3

sub-tasks (ticket-finding, accommodation-finding, attraction-finding) are posed *in sequence* as the steps to achieve trip-planning. This provides strong evidence that the accommodation-finding task is not a child of the ticket-finding task, but a sibling. The intention recognition process may provide additional support for this hypothesis by being unable to find the role of finding accommodations in the task of finding travel tickets.

Once U3 has been fully processed, the task model looks like the one depicted in Figure 4.2. The dark task box representing the attraction-finding task has the focus, since it was the last item processed from the utterance. Processing now begins for utterance U4, regarding ticket-finding. The *know-procedure* goal is adopted in response to the user's intention of teaching a procedure for ticket-finding as inferred from the utterance. Once the context has been generated for this potentially new ticket-finding task, the *remember* goal is adopted. One of the things that should be remembered is the partially elabo-

rated ticket-finding sub-task that is currently part of the task model structure. Based on the discourse structure, the system should prefer to further elaborate this existing task to generating a new task. Since this task has already been processed in response to a *learn-procedure* goal, it does not receive further processing. It does, however, receive the task model focus.

Processing now begins for utterance U5, whereby the user indicates that the system should go to Travelocity.com. Context is generated for this “going” task as a result of *generate-context* processing. The context would include available facts about Travelocity and about “going” to Travelocity. If Travelocity is recognized as a web-site, then specialized facts about navigating to web-sites can be included in the context. The process of remembering prompted by adoption of the *remember* goal might recall a learned procedure for navigating to a web-site. For the sake of illustration, assume that it doesn’t (though such a procedure would likely be present in the Expedia learned procedure mentioned previously).

The *learn-procedure*, *generate-procedure*, and *set-procedure-expectations* goals are adopted in turn. The expectations for the going task are relatively simple. The system expects that the user will take some sequence of actions which will end in “arrival” at the Travelocity destination. This expectation may include specific reference to a web navigation event if Travelocity is recognized as a web-site, or it may remain in the more general form. The *select-task-parent* goal results in further intention recognition on the goal of going to Travelocity. The current focus lies with the ticket-finding task, so it is considered as a parent first. If Travelocity is recognized as a ticket provider, then the expectation of the ticket-finding task is met, providing evidence that this is the correct place to insert the task. Or, in the tradition of Hobbs et al. ([HSAM93]), the ticket-vendor property of Travelocity could be assumed since there is no evidence to the contrary. Even if none of these are the case, there is no better expectation match for any existing task, so the currently focused task (ticket-finding) will be the attachment point for the new task (Travelocity-going). The order and extent to which each of the current tasks are

considered as potential insertion points is the question of *task/dialogue coherence*. The user will generally follow predictable patterns for communicating, which permits the use of heuristics for determining appropriate attachment points. This issue is discussed as a research question in Chapter 5. At this point in the processing, the *learn-procedure-steps* goal is adopted for the Travelocity-going event. The Travelocity-going task now has the task model focus.

The next observation that the system makes is the user's navigation to the Travelocity web-site, which is parsed into a semantic form that the system can reason about. The content of the semantic form will indicate a web-navigation event to a particular web-address. The system reasons that this is a going event which is consistent with the expectations of the Travelocity-going task. The *learn-procedure-steps* processing for the Travelocity-going task records this action and marks the task's expectations as being met. This allows a future focus shift to terminate the *learn-procedure-steps* goal.

The next utterance U6 introduces a procedure learning goal for selecting the 'Search Flights' menu entry. The context generated for this task deals with menus, menu items, and menu item selections. The expectations for this task are generated from general domain knowledge about computer interfaces, and involve the user clicking on a menu and selecting a particular entry. Next, the system determines the appropriate task parent. In addition to the focused task (Travelocity-going) already having its expectations met, the utterance indicates that the new task is to be performed sequentially after the Travelocity task. As a result, the focus is shifted upward in the task hierarchy according to the *task/dialogue coherence* principles used by the system. The parent task of ticket-finding is selected now, and its goal is consistent with the interpretation of the 'Search Flights' menu entry. The menu-selection task is therefore added to the task model as a child of the ticket-finding task and receives the task model focus. Finally, a *learn-procedure-steps* goal is adopted for this task, which handles the subsequent menu interaction. When the user selects the menu, the user interface focus shift is recorded under the menu-selection task because of coherence principles and matched expectations.

The subsequent user interface menu-selection report is also recorded, completing the expectations for the menu-selection task.

Processing the next utterance U7 adds a typing task to the model. What's interesting here is that the information 'ROCHESTER' is generated by the user without any prior dialogue reference. This indicates that 'ROCHESTER' may be a parameter to the overall trip-planning task (another *task/dialogue coherence* principle). The expectations generated for this task include a focus shift and a typing event. The ticket-finding task is selected as the task parent for the typing task according to *task/dialogue coherence* principles and the fact that the menu-selection task has its expectations fulfilled. At this point, the intention recognition component also notices that the typing event targets the FROM field, which corresponds to a ticket attribute (its point of origin). This matches with the ticket specification expectation of the ticket-finding task. The subsequent user-interface actions are recorded under this task, and its expectations are filled.

A similar process ensues for the utterance U8 regarding the trip destination. The ticket specification expectation of the ticket-finding task is matched again during the intention recognition process used to find the task parent. This multiple matching suggests that a new sub-task (ticket-specification) can be generated which has the typing tasks as children and the ticket-finding task as a parent. The resulting task model is depicted in Figure 4.5. As in the previous utterance, this utterance introduces 'ATLANTA' as new information generated by the user (and hence by the system in the future). It is marked as a possible parameter to the trip-planning task.

The remaining utterances are processed in similar fashion, and only noteworthy points in the reasoning process will be identified. Utterance U9 introduces a definite reference to the departure date, which has no referent in the dialogue up to that point. It is therefore considered a potential parameter to the task, as are the return date and time preferences from utterances U10 and U11.

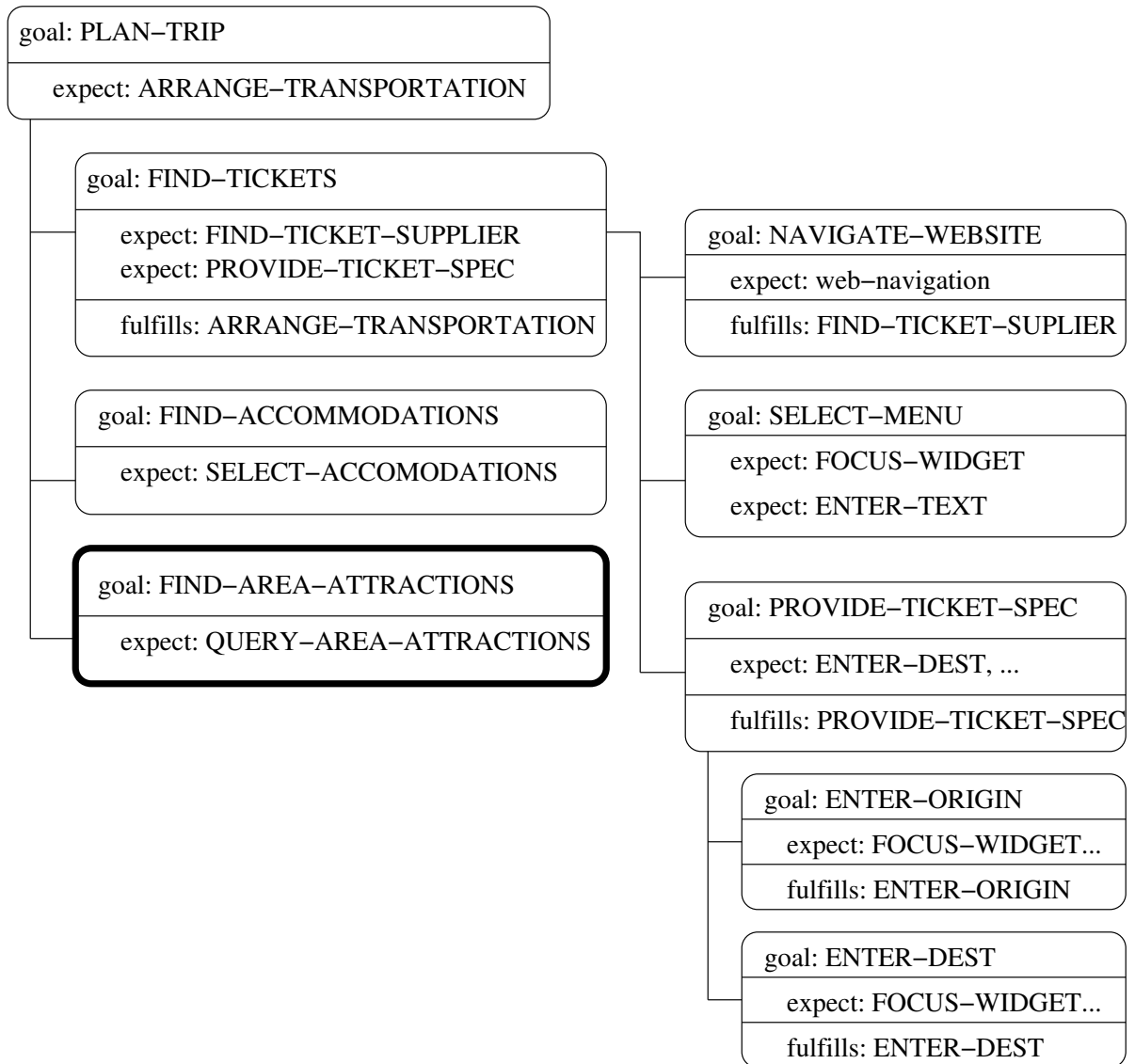


Figure 4.5: The task structure after utterance U8

Utterance U11 illustrates an interesting directive from the user. The system is told to 'do the same for the return date'. The system must infer (from a combination of intention recognition and dialogue coherence principles) that the user wishes the return date and time be specified in a manner similar to the departure date and time. Then the appropriate tasks can be generated for these actions with expectations based on the identified tasks.

Utterance U14 introduces another interesting directive from the user. The user indicates that a sequence of actions/tasks should be repeated for each flight. This requires an *inductive learning* component which can generalize from the selection of a hyperlink for a specific flight to the hyperlinks that need to be selected in order to access the information for all flights. This utterance also makes clear the need for a procedure representation that is capable of representing loops (and more generally, conditional branching). The task model outlined in this chapter ignored the details of *procedure representation*, but this is an important issue because it will determine the domains in which the task learner can operate. The example above is a relatively mundane example of the types of things that computer users do, and suggests that the ability to perform action conditionally (e.g. repeat a loop *until* a condition is met) is an important one. The issue of procedure representation is therefore an important one.

4.3 Summary

This chapter outlined a high level model to learn procedures of hierarchically structure tasks through a natural combination of user interface interaction and dialogue. In describing the model and examining its intended behavior with a simple example, I suggested a number of research problems. Constraining reasoning is important for any reasoning agent, and especially so for this model of procedure learning which includes planning and intention recognition; neither of these problems has thus far admitted efficient (polynomial time) general solutions. Heuristic methods often play a role when

the search space is too large to efficiently search. Dialogue and task coherence heuristics will provide clues for focusing the intention recognition processes in this model. The problem of indexing and retrieving procedures during learning and for task execution is important for making the results of learning available for future use (by the user and by the learning system). The inductive learning aspect of the problem (generalizing from specific instances to a general concept) is apparent in at least two guises: parameters need to be generalized from the specific instances provided in the examples, and procedures can be abstracted so that they are more useful in providing expectations for new learning. The issue of procedure representation is critical because it determines the type of learning that can be accomplished and the domains in which these procedures will be useful.

5 Research Questions

This chapter discusses some of the research questions posed by the model described in the previous chapter. These are posed as general questions, and as such, are broad categories of study. Their particular relevance to the procedure learning task should help to make study in these areas manageable.

5.1 Constraining Reasoning

The intention recognition and planning processes involved in the task learning model of Chapter 4 will require some method of inference constraint in order to remain a “soft” real-time system. The human teacher will not have the patience to teach a procedure if the inference mechanisms are not bounded. Schank ([Sch79]) proposes the concept of *interestingness* for bounding inference. Simply stated, the more interesting a concept, goal, or inference is, the more likely one is to reason about it. An inference path that is inherently interesting will be followed further than one that is not interesting. The problem with this characterization is actually distinguishing between what is interesting and what is not. Schank provides a sketch of what might be considered starting points for interestingness: the concepts of death, destruction, power, sex, money, chaos, romance, disease, etc. Though these things carry a certain amount of inherent interest, they are only interesting in certain contexts. So in addition to these concepts, he proposes that there are *operators* that make can make them truly interesting. Among these are *personal-relatedness* and *unexpectedness*. So, for instance, an unexpected death of someone closely related to a person would be of high interest to that person.

Mayfield ([May92]) uses the concept of interestingness (among other things) to limit the amount of processing performed in his PAGAN plan recognition system. A value for interestingness is defined as an inherent property of each goal that the user may pursue. As Schank noted, such a thing can be done and the assumption made that these values have been learned from experience. PAGAN also classified a user goal as interesting when it interacted negatively with the system's goals (typically that the system continue to operate properly, that no important files be overwritten, etc.). The combined interestingness of a user's inferred goal determined (in part) whether or not backchaining would continue to higher level motivating goals.

Mayfield's rather simple implementation of interestingness, however, does not seem appropriate for procedure learning. Every action that the user takes in the role of an instructor is interesting because it is part of the procedure for accomplishing the task goal.¹ The system cannot, therefore, assign relative values of interestingness to the goals that a user might pose during the course of a task. Every action ostensibly has a particular role. The notion of goal interaction is, however, relevant. If the system infers a goal for the user that is contradictory with other goals in the task, this likely indicates a misunderstanding. In these cases, it is likely easier to directly interact with the user in order to resolve the misunderstanding than to attempt to resolve it by continuing to reason out of interest.

Instead, it appears that Wilensky's ([Wil78]) criteria of stopping reasoning upon finding a thematic explanation is more appropriate for this domain. Indeed, this is exactly the role that the expectations generated by planning are intended to play. But this takes us no further toward a mechanism for constraining the intention recognition and planning components of the system.

¹There are, of course, exceptions. The user might make a mistake or specify a personal goal/task that has nothing to do with the current task (e.g. "I'm going to teach you how to buy a book. First I'll get some coffee"). Both of these situations introduce sub-goals/tasks that are unimportant to the current task, and hence somewhat uninteresting.

David Smith ([Smi89]) describes a domain-independent method for constraining backward reasoning that is based on utility. Each fact (ground literal) and each implication in a database are assigned a computed worth value. Worths provide a measure of the best outcome for inference starting with a particular database query or implication application. Inference is controlled by performing a best-first search through the inference space. For example, consider proving a particular goal G . Various inference paths might suffice, the simplest of which could be a database lookup. However, there might also be implications having consequents unifiable with G . The structure of the knowledge in the database may cause one to prefer one of these inferences to the others. The worths that are derived from the database make this decision explicit. The inference with the highest worth should be performed first. Assuming that the rule $G_1 \wedge G_2 \rightarrow G$ has the highest worth, then proving G_1 and G_2 are added as potential steps. This introduces new inferences and their associated worths. The inference with the greatest worth is applied until all choices have been exhausted or an appropriate set of goals has been proved which in combination constitute a proof for G . This method provides an excellent method for ordering the choice of steps during inference, and in fact provides the following optimality guarantee: the strategy with the lowest expected cost is derived by following steps in order of decreasing worth.

To what extent is this method applicable in the procedure learning situation? The knowledge for procedure learning does not fit the assumptions of Smith's work (knowledge is ground literals and implications, implications form a DAG). Further examination of the knowledge types and structures available in procedure learning may result in an analogy to the type of constraint that Smith's work provides for backward reasoning. Even so, there is the possibility that the reasoning performed in an optimal fashion could still be very time-consuming. Some mechanism for cutoff must be implemented.

I've attempted to include such a mechanism based on context that specifies which ele-

ments in the knowledge base are relevant for reasoning. At the time that each new task is adopted, a set of facts about the task are “activated”; these context are preferred over non-context facts. Planning, and especially intention recognition, are to be limited in their reasoning to concepts that are closely related to these context facts. In order to control how far from the context that processes are allowed to reason, each inference path is allotted an amount of mental capital that is proportional to the number of context facts that it incorporates. Each reference to knowledge outside of the context results in the spending of some of that capital, and inference is allowed to proceed until capital is exhausted. The possibility of long inference paths resulting from repeated access to context can be reduced or eliminated by enforcing an inflation mechanism (the cost of outside knowledge is proportional to the length of the inference path).

This tentative mechanism is reminiscent of Schank’s interestingness measure. A particular inference path is interesting when it has a strong correlation to the context of task expectations, and is therefore given more exploratory freedom than an uninteresting inference path with little relevance. I propose exploration of the combination of this interpretation of interestingness with the efficient ordering mechanism of Smith.

5.2 Modeling Coherence

Dialogue can add considerably to the interface for a procedure learning system. Familiar language constructs can take the place of technical user interfaces for specifying task structure and relationships, allowing novices and experts alike to train the system. Dialogue provides a more flexible user interface component, letting users provide input in a form that is natural instead of forcing users to conform their responses to sometimes cryptic and unintuitive choices. In service of this goal, dialogue systems must be designed to interact as naturally as possible, and to understand typical models of user interaction.

Dialogues in procedure learning will usually exhibit significant structure since the user is trying to teach the computer how to perform a procedure. It is clearly in the user’s best

interest to proceed logically and to clearly indicate both the task structure and transitions between sub-tasks. The extent to which each user is able to meet these criteria will vary, but most users are expected to attempt to proceed in a clear and straightforward manner.

Previous dialogue systems have operated under somewhat similar conditions and assumptions ([AP80], [Car90], [May92]). I therefore borrow the following dialogue coherence heuristics from Carberry and adapt them as appropriate. In Carberry's work, the user was seeking information for a task that would be performed at some point in the future. The system, in her case, had full knowledge of the tasks that the user was likely to query about. This is the reverse of the procedure learning scenario. In her research, the user would not necessarily ask about every aspect of a task, just those parts that they were unsure of. In this research, the user will fully explain all parts of the task model. Since Carberry's users were not executing the task at the time of interaction, there were less temporal constraints on the order in which sub-tasks were addressed. She still observed predictable interaction patterns which she recorded in the following heuristics. Any modifications predicted for the procedure learning domain will be explicitly noted.

1. If a new sub-task proposed by the user grounds in the expectations of the currently focused sub-task, then add the new sub-task as a child of the existing sub-task. This heuristic corresponds to Carberry's F1 and F2 heuristics. It models the user's propensity to fully elaborate an existing goal before proceeding on to others. Carberry justifies this principle by noting that failing to fully explore a goal when it is first introduced requires the additional overhead of re-introducing the goal in order to pursue it at a later time. Hence, the strongest expectation is that the user will continue to elaborate the most recently identified sub-task.
2. If a new sub-task proposed by the user grounds in the expectations of a sub-task along the *active path* (the path from the currently focused sub-task to the root task), then add the new sub-task as a descendant of the matching sub-task S along

the active path that is closest to the currently focused sub-task. The new sub-task should be attached to the deepest descendant of S that has matching expectations that have not yet been fulfilled. This heuristic corresponds to Carberry's F3 heuristic and models the user's preference to move on to tasks that are closely related to the sub-task most recently elaborated (accomplished by restricting S to be the closest sub-task along the active path that meets the other criteria). The heuristic has been modified slightly from Carberry's original form, which assumed that a plan for solving the goal of sub-task S would dictate exactly where to attach the new sub-task. Since the task learner does not have the same information about task structures that Carberry's system did, it is necessary to add some principle for choosing the appropriate attachment point. In this case, the deepest descendent of S with matching but unfulfilled expectations represents the most specific task that is consistent with the sub-task specification and has no associated procedure.

3. If the new sub-task has expectations into which the goals of some sub-tasks of root match, then add the new sub-task as a child of root and add the matching children of root as children of the new sub-task. This corresponds to Carberry's F4 heuristic, which is included to allow bottom up style dialogues. This form of dialogue is somewhat unexpected in procedure learning, but the heuristic is included for completeness. This heuristic description differs from Carberry's in that hers requires the root task to match the expectations of the new sub-task. The situation is different in task-learning because the top-level task should not change during the course of the learning. Some of the sub-tasks of the root task can play similar role to Carberry's root, though. The essence of this heuristic is that sometimes the user will specify tasks in a bottom-up order, addressing specific tasks before those tasks that they accomplish.
4. If the new sub-task fulfills an expectation of root that is already fulfilled by another sub-task, then add the new sub-task and the other matching sub-task as children of a new auxiliary sub-task, and add the auxiliary sub-task as a child of root. This heuristic corresponds to Carberry's F5 heuristic, which is intended to capture the

notion that some new tasks are related to the existing context via a more abstract task that has both of them as descendents. This heuristic is modified slightly from Carberry's presentation to avoid the same problem as the previous heuristic: the root task should not change during the course of learning.

In addition to the information that Carberry's system took as input, this procedure learning system will have explicit input about the task structure and the control flow. The user will likely use terms that describe the relationships between sub-tasks. Examples of the language forms that indicate such constraints are: first, next, then, lastly, in order to, because, so that, etc. This type of input places strong expectations on the location in the current task structure of the sub-task next introduced. These constraints can be used in conjunction with the principles identified by Carberry to find the appropriate point of attachment for new sub-tasks in task learning. The control flow for the task structure will also be provided primarily through natural language interaction. Complex control structures including loops and conditionals should be available to the user.

In order to verify these principles as being valid and comprehensive for the task-learning domain, a study of task-learning dialogues will be conducted. This study will seek to identify the characteristic patterns of language that users use to identify task structure, to switch between sub-tasks, and to specify control flow. In addition, it will characterize the strategies that users employ to coherently relate procedure structures, and attempt to determine the level of detail to which tasks are typically decomposed. Finally, the study will attempt to determine how well task structure can be learned without understanding the semantics of the tasks and task actions. This baseline will be useful for determining how useful the intention recognition is for learning task structure. Even if the task structure is largely retrievable from the patterns of language alone, I expect that the causal structure will benefit from intention recognition.

5.3 Inductive Learning

There are two major roles that inductive learning plays in procedure learning. The first is to recognize parameter instances in learned procedures so that they may be generalized to allow for different parameter bindings. The second is to adapt previously learned procedures while learning new procedures. Both contribute to the effectiveness of procedure learning.

The programming by instruction community has focused mainly on learning parameters from repeated task demonstration, according to Lau et al. ([LBCO04]) This body of literature (e.g. [CHK⁺93], [Lie01]) should provide a starting point for examining the issues in learning and generalizing parameters. The parameter learning that takes place for the task learning in this proposal, however, is different. Parameters need to be identified from a single training session, including the appropriate mechanisms for generalizing parameter values. A book buying task, for instance, will likely require parameters for book title and author. While these parameters might be recognized by analyzing a sequence of similar task traces where book title and author information consistently changed while other features remained static, this information is unavailable and other information sources must be used. Knowledge about the semantics of action and task roles provide the necessary information for this domain.

Knowledge about the task and the roles of related entities will help identify parameters by identifying those pieces of information provided by the user that are relevant to the task. For example, in the hypothetical book buying task, the object of the purchase is a book, making books salient for the task. When the user enters the title (i.e. “now you enter the title”), the system can associate the title being entered to the object of the purchase, based on its knowledge of book properties and features. Since the book plays a primary role in the task, and the information provided by the user (title) is a primary characteristic of the book, they system can identify the title as a potential parameter to the task. Further reasoning about purchases (e.g. that the object of purchase must

be specified to the vendor) and books (e.g. that title and author information uniquely specify a book) provides additional support for this hypothesis.

This heuristic for parameter identification will be verified, and others will be identified by analyzing task learning dialogues. This analysis will identify the types of relationships that task parameters frequently play in dialogue, and how these relationships are identified by dialogue structure. This information will guide the development of heuristics for identifying task parameters.

Probably the biggest difference between this procedure learning system and the ones discussed in the literature review (aside from the inclusion of natural language dialogue) is that this system will learn procedures from a single demonstration (and practice sessions), while most of the others require (or do a better job with) multiple instances of the same task. How can multiple procedure learning sessions be combined? While I do not propose an answer here, this is an interesting topic that I hope to address during the course of my thesis work. One step in that direction is to use previously learned procedures in the learning of new procedures.

The procedure learning system has a good deal of information available to it as it tries to learn a new procedure that is similar to one that it has already learned. The previously learned procedure will contain task structure information, task parameter information, role information, and causal relationships. The extent to which this information can be used depends on how similar the old and new tasks are, how well the structure of the old task is understood, and how much background information is available on the new task.

The case-based reasoning literature will provide a starting point for this line of inquiry. Kolodner ([Kol93]) describes a number of prototypical case-based reasoning systems and describes the fundamental principles. Adapting past experiences (cases) to new situ-

ations is one of the primary operations of a case-based reasoner, and as such should address relevant issues.

The following is a preliminary method for re-using the task structure of a previously procedure. Each sub-task from the remembered task should be reasoned about in the context of the new task to verify that it makes sense as part of a strategy for accomplishing the task goals. Those that are found to play a role should be marked as probable expectations whilst the others are left as possible expectations. For those sub-tasks that the new and old procedures have in common, the parameter role information and role relationship information can be re-used. For example, knowing a procedure for book purchasing might help one to learn a procedure for music purchasing since the steps will be very similar. Assuming that a book-purchasing procedure involving navigation to Amazon.com and pressing the “Books” button exists, a similar procedure could be hypothesized for music: navigate to Amazon.com and press the “Music” button. This type of reasoning requires that the roles for each parameter be identified. In this case, the “Books” button was related to the role “object of purchase”. The change of purchase type from books to music therefore required a change in those sub-tasks that used the “object of purchase” parameter. Causal information between sub-tasks (those that occur in both the old and new procedures) should also be transferable. It is possible, however, that certain causal relationships will be violated by changing parameter values as just described. Therefore the causal relationships that relate to a modified sub-task should also be evaluated to see if they are still applicable. Case studies are required to understand to what extent the information from a previously learned task can inform new task learnings, and how such an information transfer can take place.

5.4 Indexing and Retrieval

There are at least two times in a procedure learning system where the capability to remember previously learned procedures is useful. The first is in order to apply the learned procedure to accomplish a task at the user’s request, and the second is to use

the knowledge gained from a prior experience to inform a subsequent learning session. These, and any other uses of remembering, will require a method for remembering appropriate learned procedures.

Case-based reasoning, again, has similar requirements. Kolodner ([Kol93]) outlines the basic capabilities that a case-based reasoner requires. Among these are indexing and retrieval abilities, and particularly relevant for the second case described above is the ability to retrieve items that are similar but do not exactly match the query. The ability to purchase books encoded in a learned procedure can inform the learning of a procedure to buy music, but only if it can be recalled at the appropriate time. The case-based reasoning literature provides a philosophy for selecting indexes for cases (learned procedures in the current context), but Kolodner notes that automated systems for this task are deficient. Schank ([SA77], [Sch82]) outlines a theory of remembering based on Memory Organization Packets (MOPs) and Thematic Organization Packets (TOPs) that motivates Kolodner's approach to case-based reasoning. MOPs are structures that allow content based remembering, the type that relates a specific memory to ones that are similar in content (i.e. share similar entities, similar setting, etc.). TOPs are structures that index memories based on thematic similarities. This is a more abstract type of remembering that allows one to remember situations that are similar because of their structure and not because of their content. For instance, Schank tells the story of his daughter searching for sand-dollars in shallow water though she knows that they are more likely to be found in deeper water. When asked why she was looking there, she explained that it was easier. This reminds him of the story of a drunk looking for his keys near a street light. A passing person asks if they can help and after minutes of unfruitful searching learns that the drunk had, in fact, lost the keys in a different location but was searching near the street light because the light was better. The content of the stories is markedly different, but the structure of the goals and plans held by the story characters are similar. In both cases, the actors are applying an easy but inappropriate plan to solve a problem.

In my research on task learning, I expect that only the former type of remembering will play a major role, since the particulars of a task will have a very strong and direct affect on the actions that implement a particular procedure. The task of remembering for context learning has a more narrow scope as a result. I expect that indexing a procedure based on the primary task goal and the entities that play a role in that goal should be sufficient for the types of remembering that are needed. The ability to remember a procedure for a specific task (e.g. buying a book) should be easily located by looking for tasks that are indexed as involving buying and books. Additionally, less specific tasks can be remembered as general templates for learning by finding those tasks that involve buying and those tasks that involve books. For procedure learning, the tasks that are indexed by the major task action are more likely to provide useful results.

Is this simple indexing scheme sufficient? I intend to answer this question by applying case-based reasoning methodology for selecting case indices. This analysis will involve itemizing the types of rememberings one would expect for procedure retrieval, and determining whether they can be supported by the selected index set. Such analysis will be motivated by analyzing multiple task learning corpora to identify those places where rememberings could have informed the learning procedure, and what cues could have triggered such rememberings.

5.5 Procedure Representation

The model of the previous chapter leaves implicit the representation of the tasks and procedures that the system develops. This, however, is an important research question that must be addressed early in the development of such a system. Various content requirements (e.g. expectation information) and capability requirements (e.g. the ability to support looping and conditional execution) were described, but a complete list of such requirements has yet to be developed.

I expect that the requirements for the representation will be generated from analysis of

the expected task domains and the desired properties and capabilities of reasoning. The brief analysis of the traveling dialogue in Chapter 4 provided some initial content and capability requirements. A more comprehensive analysis of representative dialogues from the target domains will likely yield other important requirements for the task model. These requirements require support not only from the algorithms for reasoning, but also from the representation of the underlying structures. Other requirements may come from the types of reasoning that the system is called upon to perform. The ability to comparatively evaluate similar tasks, for instance, requires an analysis of task content and determination of those items that are appropriate for comparison.

The specification of a formal syntax and semantics for the representation is essential for determining whether the requirements so determined have been met. Additionally, this type of analysis makes clear the domains for which the resulting system will be suitable, and supports portability by making clear the role and function of each type of knowledge in the system.

Because knowledge underlies all reasoning activity, this task must be addressed prior to and concurrent with the other tasks identified here. An initial specification will frame the study of the other problems, which will in turn suggest refinements to the initial model.

5.6 Time line

The TRIPS system ([AFS01]), in coordination with various instrumented applications (e.g. a web browser), will provide the infrastructure for the system I've described in this proposal. This framework already provides a number of key components, whose implementation from scratch would be beyond the scope of a single thesis. Speech generation and interpretation, for instance, are components of the TRIPS architecture, and whose presence should allow focused study of the problems highlighted in this proposal. The combination of the TRIPS system with an instrumented web browser has already shown

rudimentary capabilities for task learning. A book buying task with a simple linear structure has been learned, demonstrating that the system described herein should be realizable in a few years time.

I will first address the knowledge representation issue, as it exerts influence on all of the others. This task will begin with the identification of the task domains which the system should support. Subsequently, an analysis of representative task instruction dialogues will be conducted to gather requirements for the representational capabilities of the task/procedure data structures.² If no such corpus of dialogues exists, one will be collected to support this analysis. This analysis should also yield an initial set of dialogue coherence principles and a measure of how well the proposed set of coherence principles models representative dialogues.

Once an initial set of requirements has been set forth, a suitable task/procedure representation will be developed with an accompanying semantics for its interpretation. This development will be informed by a literature review of knowledge representation treatments for similar requirements (e.g. [Fer95], [DHW94]). I expect that this analysis and knowledge representation development will be conducted over the summer of 2005.

I will next implement the high level framework described in this proposal, using as much as possible the components already present in the TRIPS architecture. I expect to focus, during this time period, on modeling dialogue coherence and constraining reasoning. These will naturally require the modification of existing TRIPS components and the generation of new ones. These problems will be addressed during the 2005/2006 academic year.

²This characterization of task/procedure representation as a data structure is premature and not to be taken literally. As Pollack has noted ([Pol90]), plans are better regarded as mental attitudes in support of plan recognition. Since this work contains plan/intention recognition as a central component, this view may be most appropriate.

Finally, I will address the inductive learning and indexing/retrieval problems described above. These can be postponed because their function is largely one that occurs after the initial task learning has occurred; a working system can be developed without them. This is not exactly true in the case of inductive learning since parameters must be learned during the initial task learning, and existing procedures can provide additional information to the learning process. The parameter generalization will be treated in the initial system development, relying heavily on the existing programming by example literature for generalization techniques. Original research will be delayed until a complete working system has been developed. These particular issues, though not treated initially, will be included in the early requirements generation process so that their inclusion is not post hoc.

5.7 Summary

This proposal introduces intention recognition into a task learning framework. The key benefits of this marriage are automatic determination of task structure (previous approaches have required annotation or have learned tasks with simple structure), automatic determination of causal relationships between structures (most systems ignore this aspect or focus on a narrow definition of causality), and the opportunity to study the interaction of multiple reasoning processes for the purpose of learning how better to integrate them. This proposal also introduces a high level model for a task learning system that incorporates intention recognition. This model is outlined by examining a learning problem that exhibits non-trivial structure.

A number of specific problems and questions are identified as relevant to this endeavor. These are the issues of knowledge representation, modeling discourse and task coherence, constraining reasoning, inductive learning, and indexing/retrieval. A basic outline of the next steps for addressing these issues is proposed along with a rough time line.

Bibliography

- [AFS01] James Allen, George Ferguson, and Amanda Stent. An architecture for more realistic conversational systems. In *Proceedings of Intelligent user Interfaces 2001 (IUI-01)*, 2001.
- [AJRS02] Richard Angros, W. Lewis Johnson, Jeff Rickel, and Andrew Scholer. Learning domain knowledge for teaching procedural skills. In *AAMAS*, pages 1372–1378, 2002.
- [All83] James Allen. Recognizing intentions from natural language utterances. *Computational Models of Discourse*, 1983.
- [AP80] James F. Allen and C. R. Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15(3):143–178, 1980.
- [Bau94] Mathias Bauer. Integrating probabilistic reasoning into plan recognition. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 620–624, Amsterdam, Netherlands, 1994. John Wiley & Sons.
- [Bra87] Michael Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts, 1987.
- [Car90] S. Carberry. *Plan Recognition in Natural Language Dialogue*. MIT Press, Cambridge, MA, 1990.
- [CG93] Eugene Charniak and Robert P. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993.

- [CHK⁺93] Allen Cypher, Danieal C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, editors. *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.
- [CL75] A. Collins and E. Loftus. A spreading-activation theory of themantic processing. *Psychological Review*, 1975.
- [CMP90] Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors. *Intentions in Communication*. The MIT Press, Cambridge, Massachusetts, 1990.
- [CSC⁺93] R. Calder, J. Smith, A. Courtemanche, J. Mar, and A. Ceranowicz. Modsa behavior simulation and control. In *Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation*, 1993.
- [DHW94] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings, 2nd Conference on AI Planning Systems*, 1994.
- [Fer95] George M. Ferguson. *Knowledge Representation and Reasoning for Mixed-Initiative Planning*. PhD thesis, University of Rochester, Rochester, New York, 1995.
- [FN71] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.
- [GRR01] Andrew Garland, Kathy Ryall, and Charles Rich. Learning Hierarchical Task Models by Defining and Refining Examples. In *First Int. Conf. on Knowledge Capture*, pages 44–51, 2001.
- [GS86] B. J. Grosz and C. L. Sidner. Attention, intention and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.
- [HSAM93] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. Martin. Interpretation as abduction. *Artificial Intelligence*, (63):69–142, 1993.
- [Kau90] Henry Kautz. A circumscriptive theory of plan recognition. In Cohen et al. [CMP90], pages 105–134.

- [Kol93] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., 1993.
- [LA90] Diane J. Litman and James F. Allen. Discourse processing and commonsense plans. In Cohen et al. [CMP90], pages 365–388.
- [LBCO04] Tessa A. Lau, Lawrence D. Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In *Intelligent User Interfaces*, pages 109–116, 2004.
- [Lie01] Henry Lieberman, editor. *Your Wish Is My Command, Programming By Example*. Morgan Kaufmann Publishers, 2001.
- [May92] James Mayfield. Controlling inference in plan recognition. *User Modeling and User-Adapted Interaction*, 2(1-2), 1992.
- [Mit82] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
- [Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [MS71] D. E. Meyer and R. W. Schvaneveldt. Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *JEP*, 1971.
- [Pol90] Martha E. Pollack. Plans as complex mental attitudes. In Cohen et al. [CMP90], pages 77–104.
- [RLR⁺02] J. Rickel, N. B. Lesh, C. Rich, C. L. Sidner, and A. Gertner. Collaborative discourse theory as a foundation for tutorial dialogue. In *Lecture Notes in Computer Science*, volume 2363, pages 542–551, 2002.
- [RS98] C. Rich and C. Sidner. Collagen: A collaboration manager for software interface agents. *User Modeling and User Adapted Interaction*, 8(3/4):315–350, 1998.
- [RSL01] C. Rich, C. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI magazine*, 22(4), 2001.

- [SA77] R. Schank and R. Abelson. *Scripts, Plans, Goals and Understanding*. Erlbaum, 1977.
- [Sch79] Roger C. Schank. Interestingness: Controlling inferences. *Artificial Intelligence*, 12:273–297, 1979.
- [Sch82] R. Schank. *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge Univ. Press, 1982.
- [Smi89] David E. Smith. Controlling backward inference. *Artificial Intelligence*, 39(2):145–208, 1989.
- [SSG78] Charles F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, (11):45–83, 1978.
- [vLL01] Michael van Lent and John E. Laird. Learning procedural knowledge through observation. In *K-CAP 2001: Proceedings of the international conference on Knowledge capture*, pages 179–186, New York, NY, USA, 2001. ACM Press.
- [Wil78] Robert Wilensky. *Understanding Goal-Based Stories*. PhD thesis, Yale, 1978.