One-Shot Procedure Learning from Instruction and Observation

Hyuckchul Jung James Allen Nathanael Chambers Lucian Galescu

Mary Swift* William Taysom

Florida Institute for Human and Machine Cognition, Pensacola, FL 32502
*Computer Science Department, University of Rochester, Rochester, NY 14627
{hjung, jallen, nchambers, lgalescu, wtaysom}@ihmc.us swift@cs.rochester.edu

Abstract

Learning tasks from a single demonstration presents a significant challenge because the observed sequence is inherently an incomplete representation of the procedure that is specific to the current situation. Observation-based machine-learning techniques are not effective without multiple examples. However, when a demonstration is accompanied by natural language explanation, the language provides a rich source of information about the relationships between the steps in the procedure and the decision-making processes that led to them. In this paper, we present a one-shot task learning system built on TRIPS, a dialogue-based collaborative problem solving system, and show how natural language understanding can be used for effective one-shot task learning.

Introduction

Our daily activities typically involve the execution of a series of tasks, and we envision personal assistant agents that can help us by performing many of these tasks on our behalf. To realize this vision, agents need to have the ability to learn task models. Researchers have attempted to learn these models by observation, creating agents that learn through observing the expert's demonstration (Lau & Weld 1999; Lent & Laird 2001). However, these techniques require observing multiple examples of the same task, and the number of required training examples increases with the complexity of the task.

One-shot learning presents a significant challenge because the observed sequence is inherently incomplete – the user only performs the steps required for the current situation. Furthermore, their decision-making processes, which reflect the control structures in the procedure, are not revealed. Natural language (NL) can alleviate these problems by identifying (i) a useful level of abstraction of observed actions; (ii) parameter dependencies; (iii) hierarchical structure; (iv) semantic relationships between the task and the items involved in the actions; and (v) natural control constructs not otherwise observable.

Sowa put it well, "Natural languages are the ultimate knowledge representation languages. They are used by everyone from infants learning their first words to scientists discussing the most advanced theories of the universe" (Sowa 2000). Encoded information in NL, as well as humans' natural ability to express it, needs to be exploited in the one-shot learning of tasks.

In this paper, we present a system that learns procedural knowledge through observation accompanied by a natural language "play-by-play". Various specialized reasoning modules in the system communicate and collaborate with each other to interpret the user's intentions, build a task model based on the interpretation, and check consistency between the learned task and a priori knowledge.

In the following sections, we introduce a motivating domain and present a task learning system built on top of a dialogue-based collaborative problem solving system, followed by its performance results on real world tasks.

Motivating Domain

While online purchasing is growing in popularity with advances in e-commerce, automating online purchasing is challenging because it consists of a series of steps within a dynamic web environment that is often populated with irrelevant information.

Figure 1 shows a sample dialogue in which a user teaches the system how to buy a book by submitting a requisition form with information that is found on a web site. After a single session, the system has learned a task model that can be executed almost flawlessly in tests with other books. The dialogue provides the high level information that is essential to learn a task:

- Task goal: line 1 shows the top-level action (i.e., purchase) and its parameter (i.e., book).
- Task hierarchy: line 8 indicates the start of a new sub task, and line 25 & 31 tell the end of a (sub) task.
- Primitive steps: utterances (e.g., line 2) describe the steps required to complete the task, and application actions (e.g., line3) are associated with the steps to realize them.
- Parameters: user utterances also provide the abstract information of the step and action parameters as well as the relation between parameters.

1. **“Let me teach you to buy a book"**
2. **"You go to the purchase form"**
3. *< Enters a URL in the browser and hits enter >*
4. **"We fill in the author field"**
5. *< Types an author into the author field >*
6. **"And the title field"**
7. *< Types a title in the title field >*
8. **"Now let me show you how to find the other information"**
9. **"Go to Amazon"**
10. *< Opens a new tab, enters a URL in the browser >*
11. **"We select the books tab"**
12. *< Clicks the tab labeled “books”>*
13. **"Then select advanced search"**
14. *< Clicks the link labeled “advanced search”>*
15. **"Put the title here"**
16. *< Types the title into the title search field >*
17. **"And put the author here"**
18. *< Types the author into the author search field >*
19. **"Then click the search button"**
20. *< Clicks the search button >*
21. **"Now we select the page for the book"**
22. *< Clicks the link with the title of the book >*
23. **“This is the price”**
24. *< Highlights the price >*
25. **"We're done here"**
26. *< Switches back to the book purchase form >*
27. **"We put the price here"**
28. *< Types the price into the price field >*
29. **"Now submit the form"**
30. *< Clicks the submit button >*
31. **"OK that’s it"**

Figure 1: Request to purchase a book

- Control structure: special wording (e.g., “if”, “until”) indicates conditionals, loops, etc.

Architecture of Task Learning System

We built a task learning system called PLOW (Procedure Learning On the Web) as an extension to TRIPS (Ferguson & Allen 1998), a dialogue-based collaborative problem solving system that has been tested in many domains.

TRIPS System

The TRIPS system provides the architecture and domain-independent capabilities for supporting mixed-initiative dialogue in a range of different applications and domains. Its central components are based on a domain independent representation, including a linguistically-based semantic form (the LF), illocutionary acts, and a collaborative problem-solving model. Domain-independence is critical for portability between domains: the system can be tailored to individual domains through an ontology mapping system that maps the domain-independent representations into the domain-specific representations.

Figure 2 shows the main components of the complete task learning system. The core components of TRIPS include (1) a toolkit for rapid development of language models for the Sphinx-III speech recognition system, (2) a robust parsing system that uses a broad coverage grammar and lexicon of spoken language, (3) an interpretation manager (IM) that provides contextual interpretation based on the current discourse context, including reference resolution, ellipsis processing and the generation of intended speech act hypotheses, (4) an ontology manager (OM) that translates between representations, and (5) a surface generator that generates system utterances from the domain-independent logical form.

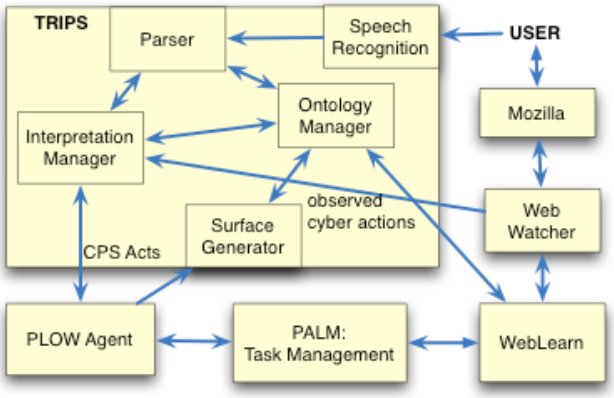

Figure 2: TRIPS architecture

The IM coordinates the interpretation of utterances and observed cyber actions, and interacts with the PLOW agent to identify the most likely intended interpretations in terms of collaborative problem solving acts (e.g., propose an action, accept, …). The IM draws from the Discourse Context module as well to help resolve ambiguities in the input, and coordinates the synchronization of the user’s utterances and observed actions. The TRIPS system has been described in detail elsewhere (Allen et al. 2002, Ferguson & Allen 1998), so we will focus the rest of the discussion on the domain-specific components that enable task learning from language.

Procedure Knowledge Learning System

The PLOW agent provides the core agent capabilities for procedure learning and execution from interactions with the user. While it is specific to the application of procedure learning, it is not specific to any particular type of procedure. It models the interaction in terms of a set of collaborative “meta” tasks that capture the actions in different modes of user interaction. PLOW maintains a stack of active tasks and is always executing the topmost task until it is completed.

The primary meta-task relevant to this paper involves learning a procedure by demonstration. It encodes the typical structure of the interactions involved in demonstrating a new task, starting with explicitly naming the procedure to be learned, then performing a sequence of actions accompanied by a spoken play-by-play to define the procedure, and completed by an explicit ending statement. It will push new tasks onto its agenda in order to

User says **"*click on the search button*"**
IM → PLOW: (commit (propose :action (choose :id c1 :object (button :assoc-with search)))
PLOW → PALM: (identify-substep :task T1 :step (choose :object (button :assoc-with search)))
PALM → PLOW: (accepted :condition (step-id S11)
<<User clicks on button B423>>
IM → PLOW: (observed (submit-form :id B423))
PLOW → PALM: (identify-example :step S11 :action (submit-form :object B423))
PALM → PLOW: (accepted ...)

Figure 3: Interaction between TRIPS modules

deal with situations that interrupt the demonstration temporarily. One such task involves clarification of a step that was not understood, and another allows the system to ignore actions while the user tries to restore an earlier state after making a mistake. PLOW can also push another learn-by-demonstration task on the stack when it recognizes that the user explicitly defines a subtask before introducing it as a step in the lower task. PLOW does not perform sophisticated intention recognition, but depends on the user signal of task initiation and termination.

PLOW uses its stack of active tasks to support the interpretation of new utterances and cyber actions by evaluating hypotheses generated from the IM. The IM uses a set of conventional speech act interpretation rules to suggest possible intended problem solving acts. It uses a set of hand-built heuristics to pick its top hypothesis and requests PLOW to evaluate its task coherence at every stage in the dialogue. If rejected by PLOW, it tries alternate hypotheses until one is found that makes sense in the current task context. Once PLOW receives input that makes sense given the current models, it uses the models to identify the reasoning actions that it will pass to PALM (Procedure Action Learning Module), which manages the reasoning required to construct the procedures. The task models also specify the content of responses that provides the basis for the system utterances.

Figure 3 shows a highly simplified trace of the messages between the IM, PLOW and PALM where the user demonstrates the next step in a procedure. IDENTIFY-SUBSTEP is a request to build a step described by the user into the task representation and IDENTIFY-EXAMPLE is a request to build an action that realizes the step. For each request, PALM sends PLOW a confirmation reply that includes the ID of a task or a step for future reference, notification of any problems, and assumptions that were made to interpret the request. The next section describes how PALM processes these requests.

To build an executable task model, agents need to understand how a human user perceives objects in an application (e.g., links/buttons/fields in a web browser) and learn how to identify them. Much of the semantic knowledge in the natural language descriptions can be used to help automate the identification of those objects that are typically described in NL. The WebLearn module learns how to identify such objects by connecting the semantic IDENTIFY-SUBSTEP descriptions with the observed action descriptions from WebWatcher, the listener that observes user behavior on our web browser.

Acquisition of Task Knowledge

A task is built from primitive steps and hierarchical subtasks. To build a task model, the system needs to not only learn each primitive step, but also the relationships between steps and the hierarchical structure of a task. This section presents the process using the dialogue in Figure 1.

Information in TRIPS is expressed in AKRL (Abstract Knowledge Representation Language), a frame-like representation that describes objects with a domain-specific ontology using cross-domain syntax. AKRL retains the aspects of natural language that must be handled by the reasoning modules. While the details of AKRL are not presented in this paper due to limited space, examples in this section show how it represents the natural language.

Task Representation for Learning and Execution

The goal of task knowledge acquisition is to construct a task model that is not only executable by agents but also applicable to further learning. To help an agent reason about executable tasks, each learned task is explicitly represented in a form that specifies the task's goals, what triggers the task, and when it is completed:

*(**task :goal** <task description>*
* **:trigger** <triggering condition>*
* **:completion-condition** (state :completed-actions*
* <list of actions required to achieve this task>)*

A new task description is created from an utterance that explicitly states what a user intends to demonstrate (e.g., "*let me teach you to buy a book*", line 1 in Figure 1). The IM informs PLOW of the user's intention, and PLOW enters into a learning mode by sending a "start-learn" request to PALM with the task information. The condition of task completion is based on the world state when a user finishes the demonstration. In a simple case, the world state can be the current ordered set of completed actions.

A task may also have a trigger: e.g., when a user says, "*let me teach you what to do when an email arrives*", the email arrival is captured as a trigger. When such an event occurs during execution, the agent invokes the task.

When a step and its actions are identified, PALM creates a rule that describes when the step should be executed. Each rule contains the precondition/name/parameters of the step, and the specific actions that realize the step:

*(**rule :precondition***
* (state **:task** <task description>*
* **:ordering-constraint** <step-ordering>*
* **:condition** <additional condition>)*
* **:step** <step description>*
* **:actions** <actions to realize this step>)*

The precondition of each rule includes the task description that is used as a cue for the step in the rule.

(**akrl-expression** :content *I29* :context ((reln *I29* :instance-of **identify-substep** :content *V285* :procedure-id P223)
(reln *V285* :instance-of **choose** :agent V291 :object *V289*) (the *V289* :instance-of **button** :assoc-with *V290*)
(the *V290* :instance-of **search**) (the V291 :equals *calo)))

Figure 4: Request to identify a step given user utterance "CLICK THE SEARCH BUTTON" (line 19, Figure 1)

(**akrl-expression** :content *I30* :context ((reln *I30* :instance-of **identify-example** :content gui292 :step-id P292 :procedure-id P223) (reln gui292 :instance-of **submit-form** :object B423)))

Figure 5: Request to identify an example for the step in Figure 4

When initiating a task, the task is pushed onto the active task stack and popped from the stack when it is finished. For each rule, the task description in the precondition is compared with the top element of the active stack. If there is a rule with a matching precondition (including the task description), the actions of the rule are executed and the world state changes, which may lead to subsequent rule firings until the task is complete. If a rule involves invoking a subtask, the ":step" part in the rule describes the subtask and no ":action" is specified. When such a rule is fired, the subtask is pushed onto the active task stack.

Learning Primitive Steps

When a user describes each step in NL, PLOW sends PALM a request to identify the step and build it into a task representation. For instance, when a user says, "*click the search button*" (line 19 in Figure 1), the utterance indicates what action the user intends to perform (a CHOOSE) and the arguments of this action (a BUTTON associated with the action SEARCH). Then, as shown in Figure 4, PLOW sends PALM a request to build the step into the task model, providing the abstract knowledge about the step in AKRL.

Given the request to identify a step, PALM checks the validity of the step based on its knowledge of the names and parameters of legal primitive steps. If valid, PALM records the step name and the value of each parameter. Each primitive step identified by a user has associated actions that realize the step. For instance, the CHOOSE step can be realized by an action of submitting a form. PALM expects the user to demonstrate this action.

When the user clicks on the search link, WebWatcher observes this action and reports it. PLOW links the observed action to the current step and sends PALM a request to learn from this example. If the action is valid for this step, PALM builds it into the representation of a primitive step. Invalid actions are detected by checking PALM's knowledge base of legal actions for each primitive step: e.g., filling a form is illegal for a step to select a link. Figure 5 shows an example in which a user performs an application specific action "submit-form" with the form ID as its parameter value.

If the user had performed this step without any verbal description, the learning system would not know the right abstraction for representing the observed actions. For instance, filling in a text field with some text may not provide enough information to infer the abstract information needed (e.g., title or author of a book). While some steps could be described in NL without real demonstration, such an approach is not always feasible. To identify a web object (e.g., link, button, etc.), pointing or highlighting is a more natural and effective method. A natural language description of the demonstration could often be unreasonably verbose, ambiguous, or technically infeasible to interpret because of the difference between the perceptual features a user refers to and the internal representation of the object.

Learning Application Object Identification

Some actions require the identification of application-specific objects (e.g., links, buttons, fields, etc. in a web page), especially when they are represented in a dynamic form (e.g., dynamic HTML files). For instance, the SUBMIT-FORM action in Figure 5 refers to a form object with the identifier B423; in future visits to the same web page, the same form object will likely have a different location on the web page.

When adding an action for a step that involves such dynamic objects, PALM inserts a special perceptual action called FIND-OBJECT into the rule representation for the step. The "in-order-to" parameter of FIND-OBJECT (in Figure 6) contains the step description that provides the semantic context of the action. By using the contextual information (i.e., the object to choose is described as a button called "search") forwarded from PALM, WebLearn can learn object identification knowledge that not only requires very little training but also provides robustness to handle changes in style or structure.

In execution, the FIND-OBJECT action asks WebLearn to find an object of interest and return its object ID. Later, PALM refers to the ID in performing the action that uses the object. To handle this dynamic object reference in the task representation, the observed object ID (e.g. B423 in Figure 5) is replaced by a variable in the rule representation: e.g., in Figure 6, the "submit-form" action has (value-of X29) where X29 stores WeLearn's return ID.

Learning Parameter Dependencies

In Figure 1, the primitive steps that are identified by the user utterances, "*we fill in the author field"* (line 4) and *"put the author here*" (line 17) have the same parameter that refers to the author of a book to buy. The learned task model should have information that the two steps share the same value. When the IM analyzes the utterances, it infers that both refer to the same object, namely the AUTHOR of B1, where B1 is the book to be bought, which was introduced in the first utterance, "*let me teach you to buy* ***a***

```
(rule :precondition (state :completed-action (…..) :task (…..))
  :step  (act :name choose :parameter ((arg :name object :value ((the V289 :instance-of button :assoc-with V287)
                                                                 (the V287 :instance-of search))))
  :actions ((act :name find-object :parameter ((arg :name type :value form)
                                               (arg :name in-order-to :value < the description of this step in AKRL >)))
                                   :returns (value :type textfield :set-to X29))
           (act :name submit-form :parameter ((arg :name object :value (value-of X29))))))
```

Figure 6: PALM task representation for the step built on the information in Figure 4 & 5

book". The same reasoning can be applied to other roles of a book such as title. That is, the proper interpretation of the knowledge already encoded in NL helps us determine parameter binding which would otherwise require numerous training examples.

Managing The Task Hierarchy

User utterances can be used as cues for the hierarchical segmentation of a task. For instance, in Figure 1, "*now let me show you how to find the other information*" (line 8) indicates that a user starts to teach a new (sub) task (i.e., finding other information), and the subsequent utterances will define this subtask. Given such interpretation, PLOW sends PALM a request to push a subtask under the current task. Then, PALM defines a new task and stores it in the task representation, adding it to the precondition of subsequent steps. When completion of a task is signaled by an utterance such as, "*we're done here*" (line 25), PLOW sends PALM a request to pop the current task and return to a pending task, if any. In this process, PALM builds a completion condition into the definition of the finished task based on the current world state. While decomposition or segmentation of a task is difficult or almost impossible to learn using observation-based techniques, the use of NL makes it feasible, moreover with a single demonstration.

Learning Control Structures

Natural language can greatly simplify learning complex control structure (e.g., conditions, iteration, loops, and recursion) that otherwise would be infeasible for some machine learning techniques and/or require multiple examples for complete learning (Lau & Weld 1999). We initially focus on a simple control structure of conditions. Assume an email task of handling a book purchase request in which a user says, "*if the subject field contains this (clicking the email subject which has the text of "book request"), then find the book information.*" Given the utterance, PLOW sends PALM a request to identify a conditional step. The request is similar to the "identify-substep" request but with additional condition information that is inserted into the precondition of a rule for the step. Learning more complex conditionals (e.g., with an "else" condition) will typically require seeing multiple examples. This is an extension we are currently developing.

Revising a Task Model

To delete a step that was just identified by a user, she/he can ask PLOW to delete it by saying, "*forget it*" or "*undo that*". Then, PLOW sends PALM a request to delete the rule for the step referenced by a previously exchanged ID. More flexible methods to revise a task model (e.g., combination of verbalization and GUI actions on a window that shows a learned task) remain as future work.

Transfer of Knowledge

Because language identifies what actions are being performed, we can relate the current action to prior instances of the same action. The language description can be parameterized so that the learned knowledge can be used for similar tasks by providing different values for the parameters. This enables transfer of knowledge to improve learning. In fact, PLOW actually learns how to find and fill in fields that the user has never described.

Furthermore, since PLOW can identify whether it already knows actions corresponding to a step, it can learn a hierarchical task involving previously learned tasks without actually seeing a complete demonstration. For instance, after teaching the system how to buy a book (Figure 1), a user may give an instruction "now you need to buy the book" in a dialogue for a different task (e.g., handling book request email). Given the instruction, PLOW checks the description of each learned task to find out whether it knows how to perform the described book purchase task. That is, PLOW checks if each task description corresponds to book purchasing. If any, PLOW pushes the task into the current task as a subtask.

Demonstration and Discussion

The PLOW system learned the procedure to buy a book shown in Figure 1 with only one demonstration, and similar dialogues have also been tested on Amazon.com and other online book retailer web sites.

To measure the impact of natural language, we evaluated the performance of a "macro-recorder" that simply recorded the web actions, ignoring user utterances. When the user selected a link (or textfield, form, etc.) in the training, it records the index of that link on the page (e.g. the 23rd link). Due to the limited capability of macro-style learning, we tested only the part of the task in Figure 1 that involves searching for a book (lines 9 ~ 22 in Figure 1), excluding the steps that fill in a requisition form. For testing, the macro acts a little differently than a basic macro in that it is parameterized with author and title.

We evaluated whether each system completed all the learned steps and found the correct book detail page for 162 randomly selected books at Amazon.com and 155

books at BarnesAndNoble.com (subset of Amazon's 162 books). The execution of the task model acquired by PLOW showed 97.5% and 96% success while the macro showed 61.3% and 36.4% respectively. The PLOW error was due to incorrect links identified by WebLearn even though the learned task model was correct.

PLOW is also compared with a commercial tool for web information extraction, AgentBuilder by Fetch Technologies, which uses syntax-based web object identification and provides a GUI demonstration interface to the user. The task models built by AgentBuilder showed similar performance – 96% and 99% success on the two sites, respectively. However, compared with PLOW, AgentBuilder requires a user to navigate multiple GUIs and wizards, provide names for schema items and connectors, and make choices from multiple radio buttons and popup menus, leading to more than five times the number of interactions to learn the same task.

In order to judge the effectiveness of PLOW with naive users, we recorded five subjects as they taught a mock-up system how to buy a book on the Amazon website. The system recorded each verbal instruction and replied with verbal acknowledgments. The users were surprisingly similar in their approaches to searching. 73 of the 95 utterances can be interpreted by the current PLOW system after very small lexicon and grammar updates. The other 22 utterances can be classified into three categories:

1. Dynamic Text: The user talks about text on the web page, creating complex noun phrases that are difficult to recognize and parse.
2. Instruction Repair: The user makes a mistake during instruction and verbally repairs the situation, such as, "*I might have spelled the author wrong*."
3. Complex Instruction: The user gives a complex description to explain his actions, such as, "*we insert an n into jeannette <sil> alright*."

Finally, PLOW is a general-purpose system that can be applied to other web tasks and can be easily extended to non-web tasks with instrumented applications that provide an API for PLOW to observe the user's actions (e.g., copy and paste in a text editor) and perform the actions. PLOW has been successfully demonstrated in other domains such as weather forecasting and e-mail handling.

Related Work

There are several existing approaches to task learning. In observation-based task learning, agents learn task models through observation of the actions performed by an expert. (Lent & Laird 2001) developed a system that learns actions and their conditions from observation traces annotated by an expert. (Lau & Weld 1999) applied the inductive logic programming approach to find actions and their pre-conditions from traces. A drawback of these approaches is that they require multiple examples, infeasible for one-shot learning. Other task learning techniques require expert annotation of selected examples as opposed to learning via observation (Garland et al. 2001).

To help a human expert provide agents with task knowledge, researchers have also developed knowledge acquisition tools such as the Tailor system (Blythe 2005) that helps a user modify the procedure that is presented in NL and reasons about the effect of the change. However, neither of these systems can learn from observation.

Conclusion

We have presented a procedure learning approach based on demonstration accompanied by a spoken "play-by-play" and showed its effectiveness in learning simple real-world tasks. The knowledge encoded in language enables agents to learn complex tasks in a more accurate, robust, and natural way. The task representation in this paper not only retains the information from a user's verbal description but also stores it in an executable and computable form. Furthermore, the acquired knowledge can be reused. The key ideas from this approach could shed more light on the usefulness of natural language in one-shot task learning.

Future work includes learning complex constructs (e.g., loop, recursion), reasoning about learned knowledge (e.g., completeness and error checking, optimization), and generalization in further learning.

Acknowledgements

This work was supported in part by DARPA grant NBCH-D-03-0010 under a subcontract from SRI International, ONR grant N000140510314, and NSF grant 5-28096.

References

Allen, J.; Blaylock, N.; and Ferguson, G. 2002. A problem solving model for collaborative agents. *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems.*

Blythe, J. 2005. Task Learning by Instruction in Tailor. *Proceedings of the International Conference on Intelligent User Interfaces.*

Ferguson, G., and Allen, J. 1998. TRIPS: An Integrated Intelligent Problem-Solving Assistant. *Proceedings of the National Conference on Artificial Intelligence.*

Garland, A.; Ryall, K.; and Rich, C. 2001. Learning Hierarchical Task Models by Defining and Refining Examples. *Proceedings of the International Conference on Knowledge Capture.*

Lau, T. and Weld, D. 1999. Programming by Demonstration: An Inductive Learning Formulation. *Proceedings of the International Conference on Intelligent User Interfaces.*

Lent, M. and Laird, J. 2001. Learning Procedural Knowledge through Observation, *Proceedings of the International Conference on Knowledge Capture.*

Sowa, J. F. 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundation*. Brooks/Cole.