Towards Situated, Interactive, Instructable Agents in a Cognitive Architecture

Shiwali Mohan and **John E. Laird**
Computer Science and Engineering
University of Michigan, Ann Arbor
{*shiwali,laird*}@*umich.edu*

Abstract

This paper discusses the challenge of designing instructable agents that can learn through interaction with a human expert. Learning through instruction is a powerful paradigm for acquiring knowledge because it limits the complexity of the learning task in a variety of ways. To support learning through instruction, the agent must be able to effectively communicate its lack of knowledge to the human, comprehend instructions, and apply them to the ongoing task. We identify some problems of concern when designing instructable agents. We propose an agent design that addresses some of these problems. We instantiate this design in the Soar cognitive architecture and analyze its capabilities on a learning task.

Introduction

We are interested in long-living, intelligent agents that demonstrate reasonably complex behavior on a variety of tasks, adapt to novel environments and operate with a certain degree of autonomy. To meet these requirements, the agents must be efficient learners, and must actively acquire new knowledge throughout their lifetimes. Learning through self-directed experience alone can be slow, requiring repeated interactions with the environment. In this paper, we investigate using instruction to greatly speed learning. The instruction we are pursuing is situated and interactive. It is situated because it takes place while the agent is performing the task, which eliminates many forms of ambiguity as the instructor refers directly to features and objects in the environment that are sensed by the agent. It is interactive because the agent can ask questions to resolve ambiguity and to gather knowledge when its own knowledge is insufficient to make progress on the task. In general, instruction is a powerful approach because it reduces the complexity of the agent's learning task in many ways.

- It reduces the perceptual complexity by focusing the attention of the learner on elements of the environment that are relevant to the task;
- It reduces the semantic complexity by identifying relevant features of objects.
- It reduces the need for exhaustive exploration by leading the learner through useful training experiences.
- It encourages discovery of dependencies between actions through explicit explanation.
- It helps the learner integrate new knowledge into a comprehensive schema through association with prior knowledge.
- It validates the newly acquired knowledge by testing performance and providing suitable reinforcement.
- It enables direct communication of desirable and undesirable states.
- It facilitates incremental learning by scaffolding.

We focus on instruction of goal-oriented behaviors in external *object-oriented*, *relational* domains, where the agent has pre-programmed primitive actions and related action models. The agent is expected to be able to learn achieve novel goals that require arbitrary compositions of its primitive actions. In this work we analyze instructable agents from three viewpoints:

1. We identify and discuss challenges of designing agents that can acquire knowledge by communicating with a human instructor.
2. We characterize the problem of learning a composite action through instruction, in an object-oriented, relational environment.
3. We present a general design that addresses some of the identified challenges and analyze agents instantiated in a cognitive architecture.

Related Work

Researchers have investigated various ways to incorporate human knowledge into agent learning; however, there has been little previous work on situated interactive instruction. There has been extensive research on learning from examples, observation, or imitation, such as early work on learning to fly a simulated aircraft (Sammut et al., 1992). In these systems (Van Lent and Laird, 1999; Allen et al., 2007; Dinerstein, Egbert, and Ventura, 2007), the agent observes a human perform the task, maps that performance onto it own capabilities and tries to induce the knowledge, goals, or reward function that the human had that produced the behavior. In contrast, with instruction, the agent is performing the task, and recieves instruction on how to perform those aspects of the tasks where it lacks knowledge.

Another approach has been to have a human instructor provide feedback on agent performance, usually within a reinforcement learning framework (Maclin and Shavlik, 1996; Thomaz, Hoffman, and Breazeal, 2006; Goertzel et al., 2008; Knox and Stone, 2010). In contrast to these works, we are interested in *interactive* and *explicit* instruction where the agent learner and instructor can communicate with each other about the task.

Recently, emphasis on understanding natural language comprehension in robots has lead to work in command driven embodied agents. This work (Dzifcak et al., 2009; Lopes and Teixeira, 2000; Lauria et al., 2001; Huang et al., 2010) has explored how natural language instruction can be used to train personal robots. However, most of this work has concentrated on the problem of grounding natural language into robots' actions and perceptions and developing embodied agents that can be driven by commands. Our work in contrast is concentrated on developing a general framework through which agents can learn procedures from commands given by an instructor and then reuse that knowledge in the future.

Some of the initial investigation of how agents can learn through explicit instruction sequence from a human expert was done by Webber and Badler (1995). The author demonstrated that agents' understanding and use of instruction can complement what they derive from their experience in the environment. Huffman and Laird (1995) identified the properties of tutorial instruction, explored the requirements on instructable agent design, and demonstrated that a simulated robotic agent could learn useful procedures from instruction in natural language. Our work is an extension to this work and looks at the general issue of designing instructable agents in a cognitive architecture.

The Instructional Learning Problem

The task of learning to act in novel environments while communicating with a human expert presents several challenging problems to agent designers. Complete or partial solutions to these problems must be developed in order to design agents that can effectively communicate their lack of understanding of the environment to a human, interpret instructions, apply them correctly to the task and acquire knowledge that is useful to the agent in future. These problems can be classified into two functional groups, the **communication** problem and the **learning** problem. In the following sections (and Figure 1), we elaborate these problems, and discuss their nature. We assume that both the instructor and the learner agent can percieve the environment and hence, their observations are similar and are grounded in the environment. In addition, we assume that the instructor can observe agent's interaction with the environment.

The *Communication* Problem

What constitutes a good dialog in an instructional setting? The goal of communication between a human instructor and an agent learner is to enhance the knowledge of the agent about the environment. In a typical setting, the agent's discourse is likely to contain information about its failure to progress in the current situation, the reason of the failure, and a question posed to the instructor. The human instructor responds by providing the information required by the agent to proceed further. The human can also ask the agent to explain its behavior, in a response to which the agent analyzes its decisions and generates a discourse that describes the reason for its behavior. The communication is *interactive*, *explicit* and *grounded* in the elements of the environment. In general, an agent capable of communicating with a human must solve three distinct but related problems. It must, compose a reasonable query (the **content** problem), generate comprehensible discourse and understand instructions (the **mapping** problem), and support a flexible dialog with the instructor (the **interaction** problem).

The *Content* Problem: *What information should be communicated to the human?* An agent acting in a complex environment is required to maintain a large state representation composed of its perceptions, internal data-structures, and semantic knowledge of the environment. If the agent is unable to apply its domain knowledge to the situation, it can communicate with the instructor to gather more information about the task. However, the complete state description is rarely relevant to the information sought by the agent. A discourse generated from its entire state would be long, hard to comprehend and would fail to convey the cause of the failure. To communicate effectively, the agent has to compose a minimal set of state elements that can help the instructor accurately identify the cause of failure and provide the correct response. Similar problems arise when the instructor asks the agent to provide an explanation of its actions. The explanation should be reasonable and at the right level of abstraction.

Several heuristics can be used to compose a precise but informative response. A model of the human instructor can be used by the agent to filter out information that cannot be perceived by a human (wireless signals, laser data, way-points) or can be deduced from observations (presence of walls in a room). Very specific information about the agents own state (voltage on motors) is also less likely to be useful. An assumption of *shared environment* can be used to limit the size of queries by referring to objects present in the environment instead of providing a detailed description of their attributes and relations.

A part of this problem - generating explanations in agents - has been analyzed by Johnson (1994) and Harbers, Meyer,

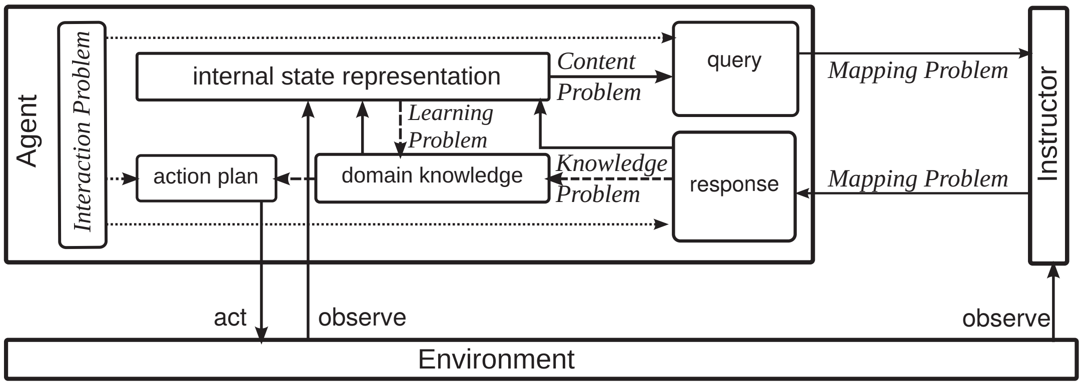

Figure 1: The Instructional Learning Problem

and Van Den Bosch (2009). The authors discuss some heuristics to filter out information to reduce information overload in explanations.

The *Mapping* Problem: *How to translate internal state representations to a commonly understood language?* Once the agent has decided the contents of its discourse, it has to translate the internal representation of the discourse to a language that can be comprehended by the human. This is the mapping problem. It also involves comprehending instructions in natural language and translating the information content of an instruction to agent's internal representations.

Solving the mapping problem in general involves complexities of natural language comprehension and generation. Just and Carpenter (1992) pointed out that instructions can be linguistically complex and difficult to comprehend, independent of the difficulty of the task being instructed. Even in simple instructions, objects and actions often are incompletely specified and agent has to rely on its domain knowledge and perceptions. Wermter et al. (2003) discuss that grounding action verbs onto agent's vision and motor actions is a hard problem, thus limiting the language that can be utilized for communication.

Goertzel et al. (2010) designed a software architecture that enables virtual agents in a virtual world carry out simple English language interactions grounded in their perceptions and actions. Other works (Dzifcak et al., 2009; Huang et al., 2010) have explored the mapping problem in embodied agents. It remains to be analyzed if these theories can applied in an instructional setting.

The *Interaction* Problem *How to communicate while acting in the environment? How to maintain a dialog?* We are interested in agents that are more than a dialog system, we are looking at agents that can maintain a dialog as they act and learn in the environment. The agent cannot direct all its resources to processing the instruction, since it has to remain reactive to the environmental changes. An instruction event is composed of several instructions that are applicable in specific contexts and order. To apply these instructions in the correct context in future, the agent has to maintain an instructional state while it is moving about and changing the environment.

The system can be constrained in several ways such that the communication is predicted and can be planned for. Communication can be *agent-initiated* only, that is, the agent begins communicating with the instructor when it faces a problem. The agent knows when to expect a response from the instructor and can process it without affecting its actions in the environment.

Allen et al. (2007) introduce a system that integrates interactive dialog with reasoning and planning. They concentrate on *instructor-initiated* communication, where the instructor begins communicating with the agent to teach it a new task.

The *Learning* Problem

How to derive knowledge from instruction and apply it in the environment? The goal of communicating with an expert is to derive new knowledge about a task. Typically, a set of instructions is provided for a new task. The agent should be able to apply an instruction within the context of the task and prior instructions. The agent should be able to merge the information contained in an instruction with its perceptions in the environment and its prior knowledge of the task such that this knowledge is applicable in future. The agent should be able to solve the following two problems.

The *Knowledge* Problem: *How to apply the instruction to the current task?* An instructable agent should be able to handle instruction events applicable to the ongoing task and discourse content. An instruction can carry a wide range of knowledge, from *semantic* knowledge about the objects in the world, *procedural* knowledge about how to navigate and handle objects, *control* knowledge about the preference order of available options to *meta* knowledge about the instruction event itself and *refinement* of pre-conditions and goal state. The agent requires the ability to identify the type of knowledge in an instructional event, relate it to the ongoing task and apply it at the appropriate situation. Given the wide range of knowledge an instruction event can carry, this is a hard requirement to meet in general. Prior systems (Webber and Badler, 1995; Huffman and Laird, 1995) constrained the instructional events to carry specific types of knowledge to allow for detailed analysis of instructional learning in agents.

The *Transfer* Problem: *How to apply the instruction to similar tasks in future?* The agent should be able to retain the instruction provided and apply it in future. However, memorizing a single instruction without a context in which it is applicable, is not useful. The agent should be able to organize the information in the complete instructional event and relate it to the current task such that it has enough information to apply the instructions in future.

The second stage of the transfer problem is the problem of learning generally applicable knowledge from instructions that apply to specific situations. Huffman and Laird (1995) introduced the concept of generalizing from *situated* explanation where the agent can acquire generally applicable knowledge by analyzing its observations of the environment when the instruction was provided. It is based on a more general theory of explanation based generalization (Mitchell, Keller, and Kedar-Cabelli, 1986).

Constraining Instructional Learning

The focus of this work is to implement and analyze agents that provide solutions to a subset of the problems identified in the previous section in a constrained learning task.

Learning Task Characterization

We are interested in analyzing agents that can use instructions to learn composite actions in *object-oriented* and *relational* domains. We define a composite action as a set of primitive actions that lead to a specific goal. The goal of a composite action is unknown to the agent before the beginning of the instruction. Instruction provides an example execution of the composite action from which the agent can deduce the goal.

The domains can be described by a set of classes, a set of predicates defined over classes, and a set of primitive actions in a classical planning setting. We define a set of classes, $C = \{C_1, ..., C_c\}$, such that each class includes a set of attributes, $Attribute(C) = \{C.a1, ..., C.aa\}$, and each attribute has a domain, $Dom(C.a)$. The environment consists of a set of objects, $O = \{o_1, ..., o_o\}$, where each object is an instance of one class, $o \varepsilon C_i$. The state of an object is a value assignment to all its attributes, $o_s = \{o.a1(Val(C_i.a1))...o.aa(Val(C_i.aa))\}$. A set of predicates, $P = \{P_1...P_p\}$ is defined over instantiated objects, $P_i(o_m, ..., o_n)$. The state of the world is the set of true predicates, $S = \{P_k, ..., P_l\}$. A primitive action, $PA = \{pa_1...pa_{pa}\}$ is characterized by pre-conditions, S_{pa} - set of true predicates for the action to apply, and a goal - set of true predicates after the action has been applied in the environment, G_{pa} and can be represented by the triplet, (S_{pa}, pa, G_{pa}).

Blocks World Domain The agents presented in this work learn composite actions in the blocks world domain (shown in Figure 3). The state can be described by two predicates: `ontop(<b1>, <b2>)` and `clear(<b>)`. It consists of two classes: `block` and `table`. Both classes have only one attribute, `name`. The domain has one primitive action: `move-block(<b1>, <b2>)`. The domain is relatively simple, however it is representative of the types of domains we are interested in. Despite its simplicity, state space is large and learning composite actions through instructions has its advantages.

Composite Actions and Learning: To learn how to perform a composite action *ca*, the agent needs to derive the following knowledge from the instruction and subsequent example execution of the task in the domain -

- *proposal knowledge*: the set of predicates that have to be true to invoke a composite action, S_{ca},
- *application knowledge*: the set of primitive actions to be executed in sequence, $\{pa_m, ...pa_n\}$, and
- *termination knowledge*: the goal conditions, G_{ca}

The composite action can be represented by the triplet, (S_{ca}, ca, G_{ca}). The number of objects $|O|$, predicates $|P|$ and primitive actions $|PA|$ in a domain give an intuition about how hard the problem of learning a new task is. The number of predicates are combinatorial in the number of objects, $|O|$ in the world, which have multiple attributes of varying values.

$$|P| = |O| \times ... \times |O|$$

The state space, S of the world is exponential in the number of predicates, $|P|$.

$$|S| = 2^{|P|}$$

Given $|S|$, determining the goal conditions for a composite action is a hard problem for learning agents. Given $|PA|$, there are many ways to reach the goal condition. Deriving application knowledge involves searching through the space of possible actions to reach the goal. If the number of primitive actions is large enough, the search space can become intractable.

Instruction can reduce some of these complexities by providing an example execution of a composite action, through which the agent can derive proposal and application knowledge and goal conditions. The sequence of primitive actions provided during the instruction greatly limits the search space and the agent can avoid exploring parts of search space which might not lead to the goal conditions. In the blocks world domain, to learn a composite action `stack(<block1>,<block2>,<block3>)`, the agent will be provided with a one grounded example of this action, `stack(A,D,C)` composed of a sequence of primitive actions starting and ending in specific states. The agent is expected to learn how to execute this specific example and to derive general knowledge about the relationship between the primitive actions and the goal state, thereby learning a general implementation of `stack(<block1>,<block2>,<block3>)`.

Instruction Event: We define an instruction event as the set of instructions provided for learning a composite action, i.e. the set of related primitive actions. For example, the instruction event for composite action `stack(A,D,C)` would include an instruction event containing the primitive actions {`move-block (D,C)`, `move-block (F,G)`, `move-block (A,D)`} (shown in Figure 3).

Problems of Interest

We are interested in analyzing two problems on the learning task previously described. We develop solutions for the **interaction** problem and the **learning** problem and make workable assumptions about others.

- The *Content* problem: To learn a composite action the agent requires the related sequence of primitive actions. We assume that the agent and the instructor know the composite action the agent is learning. To prompt for a primitive action the agent informs the instructor of the last primitive action it executed, to which the instructor responds by providing the next primitive action or indicating that the composite action has ended.
- The *Mapping* problem: We implemented a simple, limited, artificial language of communication between the agent and the instructor. The agent can map its action representation to symbols such as `move-block` and its state representation to symbols such as `ontop` and `block`. A typical sentence that describes a primitive action is `move-block moving-block D destination Table`.
- The *Interaction* problem: We focus on *agent-initiated* communication in this work. The inability of the agent to proceed further in the environment marks the beginning of communication with the instructor. The instructor only responds to prompting by the agent. (details in **Agent Implementation and Analysis** section).
- The *Knowledge* problem: We limit the information contained in the instruction to a declarative representation of the primitive action. The agent uses this information to build a declarative structure of the composite action.
- The *Transfer* problem: The agent merges the declarative structure of the composite action and context derived from the environment to derive general application and termination knowledge. (details in **Agent Implementation and Analysis** section).

Requirements on Agent Design

The properties of the learning task and the problems we are analyzing impose the following requirements on agent design. The agent should be able to -

1. represent the current state of the world.
2. represent the primitive actions and execute them.
3. model the effects of the primitive actions on the world.
4. communicate with the instructor about the primitive actions.
5. memorize the declarative structure of the composite actions and related instruction.
6. remember the context and relevant states of instruction.
7. generalize from a specific example to a general theory.

We look to Soar cognitive architecture to provide architectural solutions to these requirements.

Soar Cognitive Architecture

Soar (Laird, 2008) is a symbolic theory of cognition based on the *problem space* hypothesis, that has been extensively used in designing AI systems and cognitive models. Soar's main strength has been its ability to solve diverse problems by employing various reasoning and learning mechanisms. To represent different forms knowledge, Soar has long- and short-term memories. Following architecture elements respond to the design requirements on our agents.

Working Memory

Soar's working memory holds the agent's current state (requirement 1), which is derived from its perceptions, its beliefs about the world and knowledge in its long-term memories. The working memory is encoded as a symbolic graph structure which allows the representations of objects, relations and predicates in the world as *parent-child* relationship between graph elements. The memory is transient and changes as the agent's environment changes. The state descriptors are termed working memory elements (WMEs)

Procedural Memory

Behaviors and actions (requirements 2, 3, 4) in Soar are represented and implemented as production rules in its procedural memory. Whenever a rule's conditions match working memory structures, the rule is fired and its actions executed. An operator is the unit of deliberation in Soar and it can be either an internal or an external action. Internal actions may involve adding or removing structures from working memory thereby changing the internal state of the agent. External actions, on the other hand, initiate changes in the environment.

Procedural knowledge can be learned through Soar's chunking mechanism (Laird, Rosenbloom, and Newell, 1986). Chunking is a form of explanation based learning that captures agent's reasoning process in form of new rules which are then added to the procedural memory of the agent. Chunking also implicitly generalizes (requirement 7) by determining which state descriptors were required for the reasoning and including only them in new rules.

Semantic Memory

We use a Soar agent's semantic memory to store instructions and the task structure implicitly conveyed through them (requirement 5). Semantic memory (Derbinsky, Laird, and

Smith, 2010) provides the ability to store and retrieve declarative facts about the environment. It is context independent; it contains knowledge that is not related to when and where it was acquired. The agent can *deliberately* store parts of its working memory into semantic memory as concepts. A concept can be retrieved from the semantic memory by placing a *cue* into a special buffer in working memory. The *cue* is then used to search semantic memory for a match biased by recency and frequency and the result is then retrieved into the working memory.

Episodic Memory

Episodic memory (Derbinsky and Laird, 2009) is a context dependent memory; it records the agent's experience during its lifetime. In Soar, episodic memory includes snapshots of working memory ordered by time, providing the ability to remember the context of past experiences as well as temporal relationships between experiences. A specific episode can be retrieved by deliberately creating a cue in an episodic memory buffer in the working memory. The episodic memory searches through past episodes for the best partial match biased by recency and retrieves the episode into the episodic memory buffer. Episodic memory also provides an ability to step through episodes once an episode is retrieved. Episodic memory is used to store the context of the instruction which later is used to reason about it(requirement 6).

Decision Process

Decision-making in Soar is goal-directed. Deliberate goals in Soar take the form of operators in working memory, a distinction from other cognitive architectures where goals are often represented in declarative memory. The state of working memory causes rules to propose relevant operators. A selection mechanism makes a decision between proposed operators based on agent's selection knowledge. An operator is applied by rules that test for a set of WMEs and modify them.

However, if the operator implementation is not known directly, an impasse results and a subgoal is created to implement the operator. The operator may be implemented within the subgoal by drawing on the declarative knowledge of how to change the state or by decomposing the operator into a set of smaller operators. Through chunking, Soar compiles the reasoning performed in the substate into a new rule, effectively learning the operator implementation knowledge.

If the agent cannot make a decision between the proposed operators because it lacks selection knowledge, a tie-impasse occurs and a subgoal is created to make this decision. In this goal, knowledge from other sources such as episodic memory, semantic memory, or a look-ahead search can be used to inform the decision at the superstate. Through chunking, this knowledge can be compiled into a new rule which is used to break future ties.

A chunk - new rule created through chunking - is more general than an exact memorization of the reasoning process, due to implicit generalization and variabilization. Chunking, uses dependency analysis to determine which WMEs at the start of an impasse resulted in the final resolution of the subgoal. Since this dependency analysis uses only those WMEs that were used directly in impasse resolution, chunking generalizes implicitly. The exact situation that led to a chunk does not need to be repeated for a chunk to fire later; only those elements which lead directly to the chunk are necessary. Thus, the chunk can fire not only for the situation that created it but in any situation in which its conditions match. Chunks are variabilized; which means that the chunk refers to variables rather than exact situation and object names when possible. Therefore, the chunk can apply not only in exactly the same case, but also in analogous cases.

Agent Implementation and Analysis

In this section, we present some potential solutions to the Instructional Learning Problem introduced in the previous sections and discuss them in relation to architectural components of Soar.

Solving the Interaction Problem

As a solution to the interaction problem, we developed an instruction cycle shown in Figure 2. It is accommodates *agent-initiated* instruction and contains the following phases -

- *Detect*: The agent is not aware of any action that is applicable to the current state. It needs to pose a query prompting the instructor to provide the next action.
- *Query*: The agent identifies the state descriptors that are relevant to the current failure and stores them in a query structure. It also stores its current progress in the environment, so that it can resume functioning once the instruction is provided. It then composes an artificial language query which is presented to the instructor.
- *Parse*: The instructor responds with a complex or a primitive action that the agent should perform. The instructor response is grounded in the environment and the current task. The response is parsed and the agent identifies the objects present in the instruction and creates a representation of the action.
- *Assimilate*: The agent recalls the prior declarative knowledge it has about the task structure and related primitive actions. It records the information in the instruction such that it correctly relates to previous instructions.
- *Apply*: The agent executes the action it has been instructed to execute and observes how state progresses.
- *Explain (self)*: The agent analyzes the initial and final states of the instruction, the instructed actions and attempts to derive general knowledge about the preconditions and the goals of the action. It is an optional phase which occurs only when agent has successfully applied the composite action for the first time.

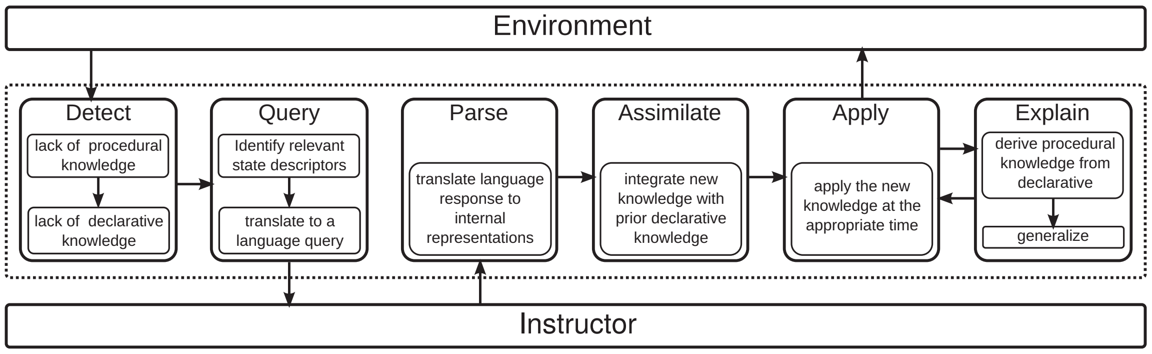
Environment
Detect
lack of procedural knowledge
lack of declarative knowledge
Query
Identify relevant state descriptors
translate to a language query
Parse
translate language response to internal representations
Assimilate
integrate new knowledge with prior declarative knowledge
Apply
apply the new knowledge at the appropriate time
Explain
derive procedural knowledge from declarative
generalize
Instructor

Figure 2: The Instruction Cycle

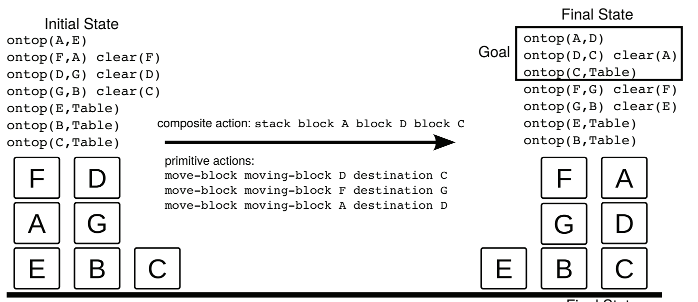

Figure 3a

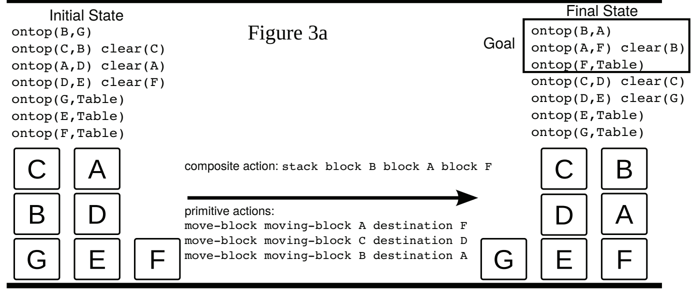

Figure 3b

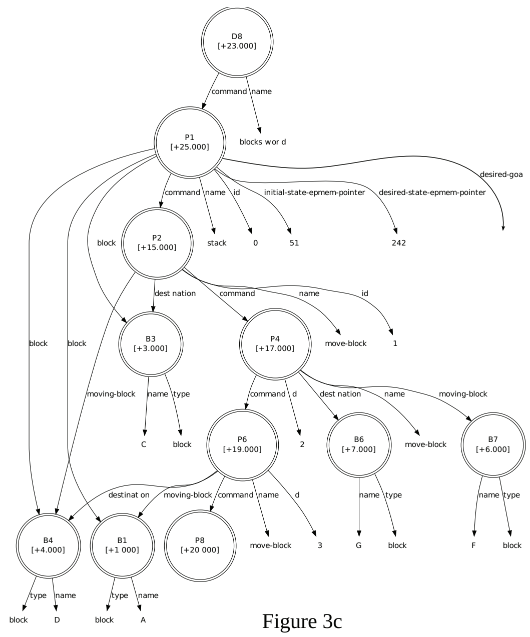

Figure 3c

Figure 3: Learning composite operator `stack block A block D block C`

Detecting Lack of Knowledge: The knowledge of the composite action in Soar agents is in two different forms - procedural and declarative. If the agent only lacks procedural knowledge, it can derive it from declarative knowledge through explanation. However, if it also lacks declarative knowledge, it must acquire it from the instructor.

- Lack of procedural knowledge: The knowledge contained in procedural memory is in the form of rules and is applied as soon as the state satisfies the preconditions. An agent 'knows' how to execute a composite action if it has rules that can *propose* the operator related to it, *apply* it by executing a set of primitive actions and *terminate* the composite action once the goal conditions are met. If any of these rules are absent, impasses will occur at various stages of the task. In response to an impasse, the agent will queries its declarative memories for information about the composite action.

- Lack of declarative knowledge: The declarative knowledge of how to perform a composite action is distributed between episodic and semantic memories of the task and is used to derive procedural knowledge. While the episodic memory stores the initial and the goal states (and intermediate states), the semantic memory stores a declarative structure of the composite action. To recall and use this knowledge, the agent must query its semantic and episodic memories. If the agent's declarative memories do not contain the required information, the related query will fail. The *detect* phase of the *instruction* cycle completes when the agent has determined that its declarative memories do not have enough information for it to use in the current task.

If the agent does not have rules that apply to the current state, and lacks related declarative knowledge in its memories, the agent queries the human instructor.

Solving the Learning Problem

Learning in our Soar agents occur in two stages - rote learning and explanation based generalization, during two runs of the same composite action. The first run of the composite action is an instructed run. The instructor takes the agent through one example of a composite action as the agent executes primitive actions in the environment. During this run, the agent acquires declarative knowledge of the specific example of the composite action. The second run is an autonomous run during which the agent derives general procedural knowledge using the declarative knowledge of the composite action. An example of the learning task (`stack block A block D block C`) in blocks world domain is shown in Figure 3.

Rote Learning: Rote learning occurs during the first, instructed run of a composite action. The agent executes the primitive actions (in Figure 3a) as it is instructed. The agent's episodic memory automatically encodes all state transitions. The agent memorizes the exact sequence of primitive actions and encodes the beginning state of the instruction as the initial state in its episodic memory. At the end of the instruction event, the final state is presented as a possible goal configuration. The instructor can generalize the goal by indicating which relations should be present in the goal state description. This semantic information about the goal state is encoded in semantic memory as the desired goal state.

Without any generalization, the agent has learned only a specific instance of the composite action through explicit memorization and does not have any knowledge of the context. This knowledge is very specific to the states it was learned in and cannot be applied to other, similar states. Figure 3c shows the declarative knowledge acquired from instruction for the specific composite action.

Situated Explanation Based Generalization: The second step is deriving procedural rules corresponding to the declarative knowledge contained in agent's semantic and episodic memories. Huffman and Laird (1995) introduced the concept of learning by *situated explanation*. We have extended this learning paradigm to include episodic and semantic memories of Soar.

Procedural learning occurs during the second run of the composite task starting from the initial state of the instruction event. During this run, instead of executing a memorized primitive action, the agent selects an operator which leads to a substate. The goal of the substate is to predict if executing the primitive action in the current state of the world leads to the goal state (retrieved from episodic memory of the first execution of the composite action). In the substate, the agent recreates the current state of the world and using the internal model of the primitive actions, it simulates changes in the state. If the prediction using such lookahead search is successful, it applies the primitive action to the current state. If the world transitions to the goal state, the agent terminates the composite action. Through chunking, this reasoning is compiled into a set of rules. These rules include the context under which these primitive actions can be applied (application knowledge). The agent also learns to terminate the composite action if it is in the goal state.

Our agent makes an assumption that the goal of a composite action is composed of only a subset of predicates that form the final state. For example, the goal of composite action `stack(A,D,C)` is the presence of predicates `ontop(A,D),ontop(D,C),ontop(C,Table)` in the state description regardless of other predicates. Learning through interactive instruction allows the agent to determine the exact composition of the goal state through explicit description by the instructor during instruction.

Due to variabilization during chunking, the new rules do not only apply to the specific object satisfying a set of predicates in the memorized instance of the composite action, but can also apply to any object that satisfies the same predicates.

Specalized goal description and variablization during chunking results in learning how to execute composite actions in states with analogous goal predicates. For example, in the second run of the composite action `stack (A,D,C)` in initial and final states shown in Figure 3a, the agent derives rules that allow it to execute `stack(B,A,F)` in analogous initial and final states without requiring more knowledge.

Learning in our current design is incomplete as the agent learns *application* and *termination* knowledge only. To complete learning the composite action it also has to acquire *proposal* knowledge. The issue of learning proposal knowledge will be explored in future.

Conclusions and Future Work

Learning through *interactive*, *explicit* instruction is a powerful learning mechanism and can be used for learning composite actions in semantically complex domains. We identified general problems that arise while designing agents that can learn from human instructions and discussed some of the solutions to these problems developed by the AI research community. We demonstrated that learning a composite action by exploration only is an extremely hard problem because the goal of the action is unknown to the agent. Explicit sequence of instructions from an expert can help the agent identify the goal state, thus reducing the complexity of the learning task. Explicit sequence of instructions can also help constrain the search space of primitive actions to reach the goal.

To solve some of the problems identified, we proposed the *instruction cycle* that supports *agent-initiated* communication. We instantiated agents based on this design in Soar cognitive architecture. We show that using Soar's declarative and procedural memories and related learning mechanisms, agents can learn composite actions by *rote* and *situated explanation*.

There are several interesting venues for future work. We are interested in studying the Instructional Learning Problem in a greater detail and understand the computational challenges involved in solving these problems. We are also looking at understanding different kinds of instructional events that contain varying kinds of knowledge, and how these instructions can be accommodated in a general design.

Acknowledgments

This research was funded by a grant from US Army TARDEC.

References

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Demonstration of PLOW: A Dialogue System for One-Shot Task Learning. In *Proceedings of Human Language Technologies*.

Derbinsky, N., and Laird, J. 2009. Efficiently Implementing Episodic Memory. In *Proceedings of the 8th International Conference on Case-Based Reasoning*.

Derbinsky, N.; Laird, J.; and Smith, B. 2010. Towards Efficiently Supporting Large Symbolic Declarative Memories. In *Proceedings of the 9th International Conference on Cognitive Modelling*.

Dinerstein, J.; Egbert, P. K.; and Ventura, D. 2007. Learning Policies for Embodied Virtual Agents Through Demonstration. In *In Proceedings of International Joint Conference on Artificial Intelligence*.

Dzifcak, J.; Scheutz, M.; Baral, C.; and Schermerhorn, P. 2009. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *IEEE International Conference on Robotics and Automation*.

Goertzel, B.; Pennachin, C.; Geissweiller, N.; Looks, M.; Senna, A.; Silva, W.; Heljakka, A.; and Lopes, C. 2008. An Integrative Methodology for Teaching Embodied Non-Linguistic Agents , Applied to Virtual Animals in Second Life. In *Artificial General Intelligence*.

Goertzel, B.; Pennachin, C.; Araujo, S.; Silva, F.; Queiroz, M.; Lian, R.; Silva, W.; Ross, M.; Vepstas, L.; and Senna, A. 2010. A General Intelligence Oriented Architecture for Embodied Natural Language Processing. In *Proceedings of the 3d Conference on Artificial General Intelligence*.

Harbers, M.; Meyer, J.; and Van Den Bosch, K. 2009. Explaining Simulations Through Self Explaining Agents. *Journal of Artificial Societies and Social Simulation* 12(3).

Huang, A. S.; Tellex, S.; Bachrach, A.; Kollar, T.; Roy, D.; and Roy, N. 2010. Natural Language Command of an Autonomous Micro-Air Vehicle. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Huffman, S. B., and Laird, J. E. 1995. Flexibly Instructable Agents. *Journal of Artificial Intelligence Research* 3.

Johnson, W. 1994. Agents that Learn to Explain Themselves. *Proceedings of the Twelfth National Conference on Artificial Intelligence*.

Knox, W. B., and Stone, P. 2010. Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. In *In Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*.

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in Soar: The Anatomy of a General Learning Mechanism. *Machine Learning* 1(1):11–46.

Laird, J. 2008. Extending the Soar Cognitive Architecture. In *Proceeding of the Conference on Artificial General Intelligence*.

Lauria, S.; Bugmann, G.; Kyriacou, T.; Bos, J.; and Klein, A. 2001. Training Personal Robots Using Natural Language Instruction. *Intelligent Systems, IEEE* 16(5):38–45.

Lopes, L., and Teixeira, A. 2000. Human-robot interaction through spoken language dialogue. In *Proceedings of International Conference on Intelligent Robots and Systems*.

Maclin, R., and Shavlik, J. W. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22(1-3):251–281.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1(1):47–80.

Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In *Proceedings of the Ninth International Workshop on Machine Learning*.

Thomaz, A.; Hoffman, G.; and Breazeal, C. 2006. Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 352–357.

Van Lent, M., and Laird, J. 1999. Learning Hierarchical Performance Knowledge by Observation. In *In Proceedings of the 16th International Conference on Machine Learning*.

Webber, B., and Badler, N. 1995. Instructions, Intentions and Expectations. *Artificial Intelligence* 73(1-2):253–269.

Wermter, S.; Elshaw, M.; Weber, C.; Panchev, C.; and Erwin, H. 2003. Towards Integrating Learning by Demonstration and Learning by Instruction in a Multimodal Robot. In *Proceedings of the IROS-2003 Workshop on Robot Learning by Demonstration*.