Cognitive Robotics using the Soar Cognitive Architecture

John E. Laird, Keegan R. Kinkade, Shiwali Mohan, Joseph Z. Xu
Computer Science and Engineering
University of Michigan, Ann Arbor, MI 48109-2121
{laird, kinkadek, shiwali, jzxu}@umich.edu

Abstract

Our long-term goal is to develop autonomous robotic systems that have the cognitive abilities of humans, including communication, coordination, adapting to novel situations, and learning through experience. Our approach rests on the integration of the Soar cognitive architecture with both virtual and physical robotic systems. Soar has been used to develop a wide variety of knowledge-rich agents for complex virtual environments, including distributed training environments and interactive computer games. For development and testing in robotic virtual environments, Soar interfaces to a variety of robotic simulators and a simple mobile robot. We have recently made significant extensions to Soar that add new memories and new non-symbolic reasoning to Soar's original symbolic processing, which improves Soar abilities for control of robots. These extensions include mental imagery, episodic and semantic memory, reinforcement learning, and continuous model learning. This paper presents research in mobile robotics, relational and continuous model learning, and learning by situated, interactive instruction.

Introduction

Our goal is to support the creation of robotic agents that have the cognitive capabilities of humans. General intelligent entities are distinguished by their ability to pursue a wide variety of goals embedded in many different problem spaces and to use large bodies of different types of knowledge in many ways – to assess the current situation in the environment, to react to changes in the environment, to deliberately select actions in order to pursue goals, to plan future actions, to predict future states, to reflect on past behavior in order to improve future performance, and to adapt to regularities in the environment, all in real time. Approaches such as MDPs work well for specific tasks where there are limited and predictable numbers of features; however, they do not scale to complex behavior and planning; nor do they address how an agent efficiently manages its memory of situations, events, and the structured knowledge it acquires through experience.

Our approach is to develop adaptive robotic agents using a cognitive architecture (Langley, Laird, Rogers 2009) that is integrated with perceptual and motor systems. The cognitive architecture provides the underlying knowledge representations; memories to hold both short-term and long-term knowledge; processes for decision making, accessing memory, and learning; and interfaces that support the transductions of continuous perception into symbolic structures and discrete motor commands into continuous control systems. The same architecture is used across all tasks, as is task-independent knowledge that supports general capabilities such as planning. Domain knowledge specializes behavior to specific tasks so that a single agent, encoded with knowledge for many domains, can pursue a variety of tasks and with learning, an agent can dynamically extend the tasks it performs. Each new task or domain does not require starting from scratch – task knowledge must be added, but the architecture and general knowledge is shared across all tasks. A cognitive architecture also provides the focal point for integration of research on perception, control, motor systems, and cognition. Research into more advanced capabilities, such as coordination, metacognition, and advanced learning mechanisms builds on those core capabilities.

We use the Soar cognitive architecture (Laird 2012) for our research in cognitive robotics. Soar was the first cognitive architecture integrated with real robots and has been used in multiple robotic agents (Laird et al. 1991; Laird & Rosenbloom 2000; Benjamin, Lyons, & Lonsdale 2004; Benjamin, Lonsdale, & Lyons 2006; 2007). Recently, we made significant extensions to Soar, many of which enhance Soar's ability to support cognitive robots.

In this paper, we start with an overview of Soar, emphasizing the recent extensions to Soar and how the cognitive components of Soar interface to sensors and motor systems. We then present some of our current research projects where we are expanding the abilities of Soar robotic agents. We start with a Soar agent that controls a small mobile robot and performs missions in a combined simulated/real world domain. This agent demonstrates how many of the mechanisms in Soar provide capabilities for cognitive robotics. We then describe the integration of SLAM so that the agent builds an internal model of the environment from laser-range data. The following section describes research on learning continuous and relational models from experience. Finally

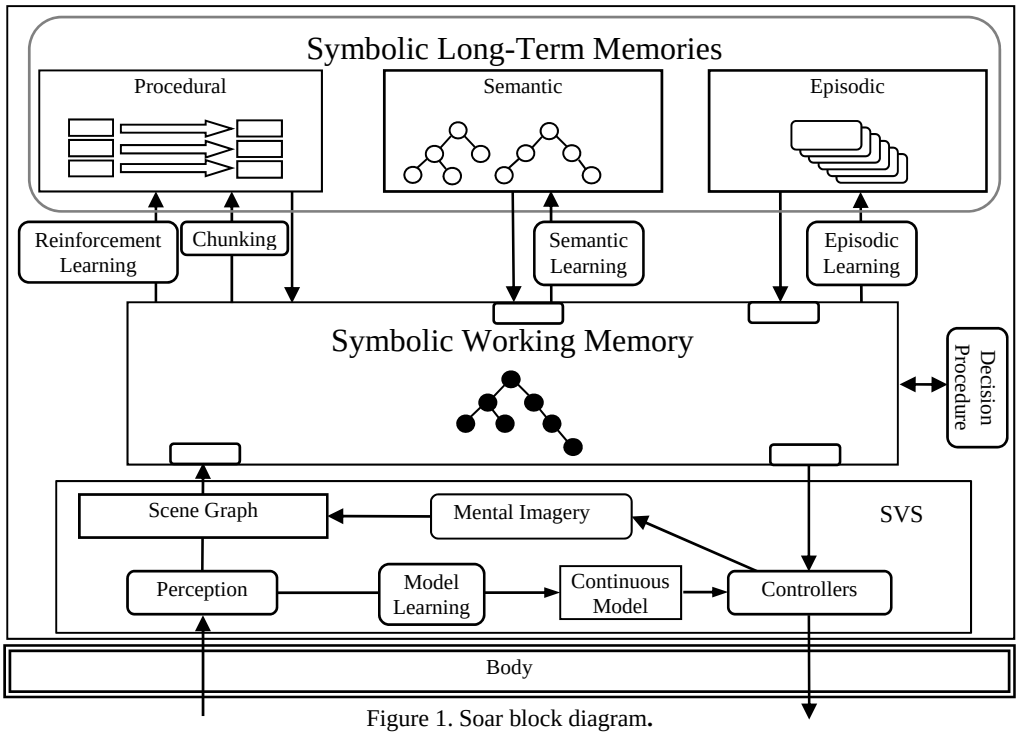

Figure 1. Soar block diagram.

we describe the use of situated interactive instruction, where a human can use language to dynamically extend an agent's knowledge and skills through instruction.

Basics of the Soar Cognitive Architecture

Figure 1 shows a structural view of Soar in terms of its primitive memories (square-edged modules), processes (round-edged modules), and their connections (arrows). Near the bottom of the figure is Soar's task-independent Spatial Visual System (SVS; Lathrop, Wintermute, & Laird 2011) that supports interaction between with the continuous representations required for perception and motor control and the symbolic, relational representations in Soar. The continuous environment state is represented in SVS as a scene graph composed of discrete objects and their continuous properties. The identity of objects and extracted symbolic features are added to symbolic working memory; however, spatial relations are extracted only as requested: the Soar agent can query the scene graph with binary spatial predicates such as "left-of(A, B)" or "contacting(C, D)". Thus, the set of predicates is task-independent and fixed in the architecture, but decisions about which predicates to extract are determined by task-specific agent knowledge.

In additional to extracting a symbolic description of the environment, SVS transduces discrete actions into continuous control of the motor system. SVS not only supports action in the external environment, but also action on its internal representation of perception. This allows an agent to use *mental imagery* (Kosslyn, Thompson & Ganis 2006; Wintermute, 2009) where the agent can perform hypothetical reasoning on a continuous representation. The continuous controllers can support arbitrary behavior, such as nonholonomic control of a car, using modern path planning algorithms, such as RRT (Wintermute 2009). Planning at the continuous level uses models of continuous environment dynamics learned online and incrementally (Xu & Laird 2011). These capabilities overcome one of the major weaknesses of pure symbolic systems that are prisoners to their symbolic abstractions. Soar can reason/plan using both symbolic (described below) and non-symbolic structures. When planning with the non-symbolic structures, the agent can monitor and control the non-symbolic reasoning by extracting relevant symbolic predicates. This approach has proved successful across many complex, continuous domains where pure symbolic reasoning is inadequate (Wintermute 2010).

On the symbolic side, working memory maintains relational representations of current and recent sensory data, current goals, and the agent's interpretation of the situation given its goals. Working memory buffers provide interfaces to Soar's long-term memories and motor system.

Semantic memory is Soar's permanent store of its global world model, and corresponds to the long-term declarative memory in ACT-R (Anderson et al. 2004). Knowledge is retrieved from semantic memory via deliberate associative retrieval. Although knowledge accumulates in this memory as new information is discovered, it can be initialized from existing knowledge bases. Soar also has an episodic memory that stores snapshots of the agent's experiences. Episodic memory provides an agent with the ability to reflect on prior experiences and use them to guide future behavior and learning. Episodic memory is often overlooked in cognitive systems, but is critical for

providing access to contextualized experiences. Most robot architectures ignore semantic or episodic knowledge, and have only limited, task-dependent long-term declarative memories. Agents developed in these architectures are only in the "here and now" and do not have the sense of history made possible by these additional memories.

Procedural memory contains Soar's knowledge of how to select and perform discrete actions, encoded as if-then rules called productions. Productions fire in parallel whenever they match working memory and act as an associational memory for procedural knowledge. Productions support the proposal, evaluation, selection, and application of *operators*. Operators are the locus of decision making in Soar. Operators are proposed (created in working memory) by rules based on the current context. Additional rules evaluate the proposed operators, creating *preferences*. This evaluation knowledge can be tuned by reinforcement learning if reward is available, either from the environment or from internal sources (Marinier, Laird & Lewis 2008). The preferences are analyzed by a fixed decision procedure and if a single operator is unambiguously preferred by the evaluation knowledge, it is selected for the cycle. If not, an impasse occurs and a substate is created in which more deliberate reasoning can occur, including task decomposition, planning and search methods.

Once an operator is selected, rules sensitive to its selection perform its actions by modifying working memory. An action can be an internal reasoning step, a query to SVS, a retrieval from episodic or semantic memory, or an external motor command. Operators in Soar correspond conceptually to STRIPS operators; however, in Soar, the component parts of operators (preconditions and actions) are decomposed into individual rules, providing disjunction preconditions and disjunctive conditional actions. In addition, Soar has task-dependent rules for operator evaluation, providing context-dependent and fine-grain control of operator selection and application.

In Soar, complex behavior arises not from complex, preprogrammed plans or sequential procedural knowledge, but instead from the interplay of the agent's knowledge and the dynamics of the environment on each primitive cycle. It is the speed of that cycle that determines agent reactivity. Based on our experience and informed by the human cognitive system, the maximum time per cycle must be below 50-100 msec. for an agent to maintain reactivity. Unfortunately, in most cognitive architectures the basic cognitive cycle can be much longer, which greatly complicates their integration with real time perceptual and motor systems. In contrast, Soar's cycle is consistently below 50 msec. and averages below 1 msec. on commodity computer hardware (Laird et al. 2010).

Mobile Robot Control in Soar

In this section, we describe how Soar controls a small custom-built robot that uses a mixture of real-world and virtual sensors to navigate an office building. The robot

Figure 2. Partial trace of agent exploration.

can move forward and backward, and turn in place. It has a LIDAR mounted in the front that gives distances to 180 points throughout 180 degrees that it uses for obstacle avoidance. The robot can sense its own location based on odometry, and it can also sense the room it is in, the location of doors and walls, and different types of objects. These additional sensors are simulated, and when controlling the real robot, the agent receives a synthesis of real and simulated sensor data. During a run, there are approximately 150 sensory data elements. The Soar agent interfaces to the robot through the Lightweight Communications and Marshalling (LCM) library for message passing (Huang, Olson, & Moore, 2010).

The evaluation described below uses a simulation because of the difficulties in running long experiments in large physical spaces. The simulation is quite accurate and the Soar agent used in the simulation is *exactly* the same as the agent used to control the real robot.

The agent performs multiple tasks, including exploring, cleaning rooms, and patrolling. A typical mission will involve initial exploration, followed by room cleaning, and then patrolling once all rooms are cleaned. Figure 2 shows a partial trace of the agent's explorations at the beginning of a mission. In room cleaning, the agent picks up and moves specific types of blocks (such as square green blocks), one by one, to a storage room. The agent dynamically constructs a map of the rooms, doorways, and their spatial and topological layout, as well as the locations of all blocks, as it encounters them during exploration.

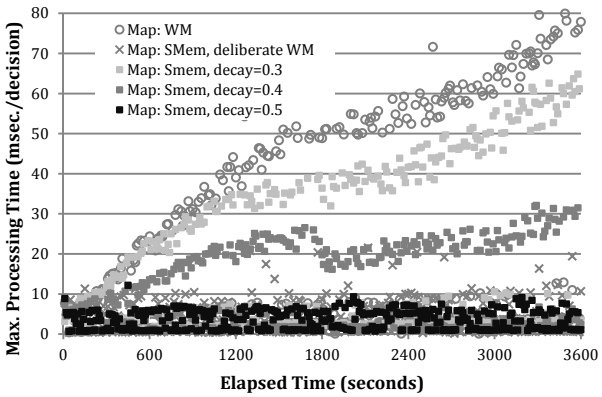

Figure 3. Maximum processing time per decision for agents that explore the map in Figure 2 using different memory

The agent uses dynamic task decomposition and abstraction planning as needed. For example, when attempting to clean up by moving a block to the storage room, the agent internally searches for a path to the storage room using the map knowledge the agent has built up through experience. The search is a variant of A*. The internal search is performed within the agent, using rules, spread across multiple cycles. Over time, planning is converted into reactive selection knowledge through Soar's chunking learning mechanism.

As part of our research, we have investigated the costs and tradeoffs of maintaining topological map information in working memory versus semantic memory, as well as using episodic memory for accessing historical data, such as the most recent locations of specific objects (such as when was the last time the agent saw a blue block). One issue is that it is easier and faster to access map data if it is maintained in working memory instead of semantic memory because semantic memory requires explicit retrievals. However, the time to access and reconstruct prior episodes from episodic memory increases as working memory grows in size. Thus, there are tradeoffs in maintaining the map in working memory versus semantic memory.

Figure 3 shows the results of an experiment where the robot explored the map in Figure 2, periodically accessing episodic memory to retrieve historical information. The x axis is time in seconds, where the agent runs for one hour (3600 seconds). The y axis is the maximum time for a processing cycle in msec. There are five sets of data. The first is from maintaining the map in working memory. This shows significant slowdown because of the growth in working memory and its interaction with episodic memory. The second is from maintaining the map in semantic memory but including task dependent knowledge to deliberately remove duplicate map information, which greatly decreases the average size of working memory and maintains reactivity. The next three data sets are agents where the map is also maintained in semantic memory, but where elements in working memory are automatically removed based on activation, which decays over time. The data sets differ in terms of the rate of decay. The graph shows that with a decay rate of .5, the automatic mechanism performs as well or better than the deliberate mechanism (decay rates higher than .5 cause the agent to thrash as it is unable to maintain sufficient internal state for planning).

Extending Mobile Robot Control with SLAM

In the previous section, the agent used its real sensory system only for obstacle avoidance, and depended upon a simulation to provide it with symbolic descriptions of its local topography. While ground truth data may be known within simulation, real world operation requires algorithms capable of processing the inherent error found within sensor measurements. In this section, we describe how we extended the agent by incorporating simultaneous localization and mapping (SLAM) into the agent, so that the agent builds up its own representations of rooms and doorways from scratch. Although one could imagine incorporating SLAM into SVS, in this case, a separate module builds up the appropriate data and then feeds it to working memory. Soar maintains control of the robot, and the SLAM process runs in the background as the robot moves through its environment.

Our mapping system uses wheel encoders to provide movement information, and LIDAR to provide environmental information. In order to deal with measurement error inherent in both sensors, a solution to the least-squares SLAM formulation known as square root smoothing and mapping (SAM) is employed (Dellaert & Kaess, 2006). This algorithm provides the system with a state estimation of the robot's movement throughout the environment. The sensor observations also provide the necessary data to detect gateways adjoining unique areas. Using the SLAM state estimation, a better approximation of the location of gateways can be retrieved and added to the SLAM map for future gateway data association. Additional algorithms were developed for finding doorways. Together, these algorithms allow for a topological representation of the environment to be built incrementally. A key feature of the algorithms is that they build up symbolic, topological descriptions of rooms and doorways, and are able to detect when two doorways are in fact the same one, just sensed from different rooms.

Figure 4 show a trace of the agent as it explores a test environment generated via solving the SLAM problem. The poses of the robot (red) are connected via constraint edges (green) recovered through LIDAR scan matching. The ground truth movement of the robot is shown in yellow, whereas the raw odometry movement is in black.
A detailed description of the underlying implementation is beyond the scope of this paper and available from Kinkade (2012). The software is based in part on the April Robotics toolkit provided by Edwin Olson.

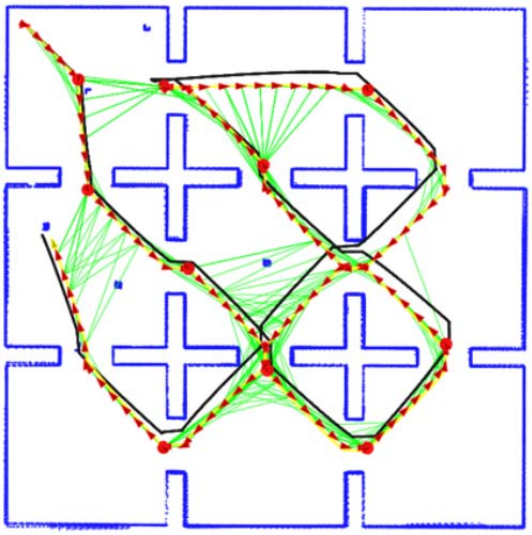

Figure 4. Trace of robot exploring rooms using SLAM.

This integration demonstrates how it is possible to use special-purpose sensor processing systems with Soar. Future work includes creating a tighter integration with SVS as well as investigating how map information maintained in the SLAM system can be more directly shared with the map representations in Soar.

Learning Continuous and Relational Models

In the robot domain described in the previous section, the agent was able to perform its tasks effectively by moving toward landmark values in its continuous state representation, such as when its relative bearing to a waypoint was close to 0.0, and using symbolic actions such as move forward and turn left/right. The agent's procedural knowledge implicitly encoded a simple model of how the continuous state properties change under a certain action, e.g. turning left will lower the relative bearing to the waypoint. A more general agent must be able to perform in novel continuous domains where actions are continuous rather than discrete, and where the agent has no a priori knowledge of how these continuous actions change the state. In these cases, the agent can speed learning and improve the robustness of its behavior by learning a model of how continuous state properties change with continuous actions. We call these *continuous models*.

The success of the robot agent was also dependent upon its ability to plan at the abstract level of rooms and navigating between rooms. This abstraction reduced the plan space by orders of magnitude from the level of turning and driving. Furthermore, casting the world into symbolic relational structures allowed the agent to employ universal weak methods such as look-ahead search and means-ends analysis to solve the problem. Being able to reason at the abstract level requires not only a mapping from concrete continuous states into abstract symbolic states, but also a model of how the abstract state changes in response to actions. We call these *relational models*.

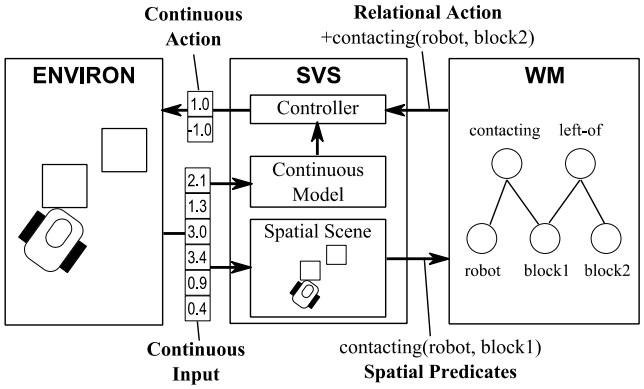

Figure 5. Overview of models and abstraction.

In this section, we describe how SVS learns continuous and relational models, how continuous models are used by a controller for generating continuous plans that implement relational actions, and how these components allow the agent to automatically learn and plan in relational abstractions. Figure 5 gives an overview of this system.

Continuous Model Learning

Formally, the continuous model is a function $f(x,u) \rightarrow y$, where x and y vectors of real numbers representing the continuous properties of the starting and resultant states are encoded by the SVS scene graph, and u is the continuous action. In the robot domain, the state vectors include the (x, y, z) coordinates of all objects as well as the robot's velocity and rotation rates of the wheels. We assume that all properties that play a role in environment dynamics are observable (there is no hidden state). The robot's continuous action u is a pair of real-numbered motor voltages to the left and right wheels.

In the object-oriented spatial environments that SVS was designed for, objects often exhibit qualitatively distinct behaviors based on their interactions with other objects. For example, a ball flying through the air follows a parabola, but upon contacting the ground it abruptly changes to a bouncing trajectory. The continuous model our system learns is composed of a set of linear functions and a classifier that determines which function to use for a particular state. The linear functions capture the individual qualitative behaviors, while the classifier determines which behavior is being exhibited. Figure 6 depicts the prediction process. To make a prediction starting in state x and taking action u (❶), the model first calculates a spatial predicate description p (❷) of the starting state x. p is a vector of the values of all spatial predicates that can describe the

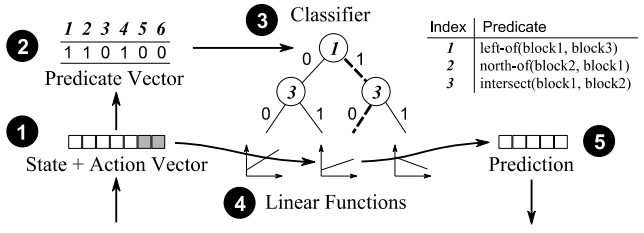

Figure 6. Prediction with continuous model.

environment. A decision tree (❸) in which the nodes test the values of single predicate classifies p into one of several qualitative behaviors. For example, states where a ball is in free-fall may be classified under one behavior, states where the ball is contacting the ground under another. Next, the linear function (❹) associated with the behavior is used to predict precisely how x will change in response to u (❺). This two-level approach is similar to several existing systems (Quinlan 1992, Toussaint & Vijayakumar 2005). The important difference is that classification is based on position-independent spatial relationships rather than splitting the input space into contiguous regions, leading to more general models.

Model learning occurs in the opposite order. Given a training instance (x, u, y), the model learner first uses Expectation-Maximization to simultaneously determine which qualitative behavior generated the instance and also learn the parameters of the associated linear function. Once the instance is assigned to behavior q, the predicate description p of x is calculated. The pair (p, q) is then used as a training instance for the decision tree classifier.

Relational Abstraction and Model Learning

Previously, we had described how the symbolic agent can query SVS's scene graph for the truth values of spatial predicates such as "left-of(robot, block1)." This is the primary method of abstracting from continuous to relational state.

For a complete abstraction, the agent must be able to perform relational actions that can change the state. We define these relational actions to be changes to single predicate values, such as "Make contacting(A, B) true." SVS has a controller module that operationalizes these relational actions into trajectories of continuous actions sent to the environment. For example, when the agent performs the relational action "Make contacting(robot, block1) true," the controller will use greedy or random-tree search techniques to find a sequences of motor voltages that will drive the robot to block1. The controller uses the learned continuous models when performing these look-ahead searches. Note that although relational actions are associated with single predicate changes, additional predicates may change as side effects, or the controller may fail to find a suitable trajectory, and no predicates will change at all. For example, executing "Make contacting(robot, block1) true" may result in making contacting(robot, block2) false.

With the abstraction in place, the agent can learn a model of how the relational state changes in response to relational actions, allowing it to plan completely at the relational level. Our system uses an instance-based learning method to capture these behaviors. For every decision cycle in which at least one predicate changes value, the relational state is stored as an episode in episodic memory. To make a prediction for a starting relational state s and relational action a, the agent queries episodic memory for an episode that is most similar to s in which it also performed a. Similarity is measured as the degree to which the predicate structures of two relational states align, discounting object identities. The predicate changes that were recorded in episodic memory for the retrieved state are then mapped into the starting state to form a prediction. Note that because similarity is based on relational structure rather than object identity, the training episodes generalize to novel states using a form of analogical mapping. This work is described in more detail in Xu & Laird (2010).

To summarize, SVS provides a symbolic representation of the continuous environment state, and grounds desired predicate changes into continuous action trajectories. The continuous model learner learns how continuous actions change the continuous state. This model allows the controller to plan continuous trajectories that implement relational actions. The relational model learner learns how relational actions affect the abstract relational state, allowing the agent to create abstract plans. These methods enable Soar agents to autonomously transition from having minimal domain knowledge to being able to plan with relational abstractions in novel continuous domains.

Situated Interactive Instruction

Our final section addresses the issue of how a robotic agent can learn from interaction with a human. Much of the prior research on robot learning from human interaction has focused on learning from demonstration, imitation, and observation. Although these methods can be useful, they are not efficient for communicating the hierarchical goal structures that are often needed for complex behavior.

Our approach is to focus on natural language instruction (Huffman & Laird, 1995), where the agent is actively trying to perform the associated task while receiving instruction. The approach we are pursuing has the following characteristics:

1. *Situated*: Instructions are grounded in particular situations, with specific objects, eliminating many forms of ambiguity. The instructor does not need to come up with general rules or procedures.
2. *General*: The instruction mechanism is general and can be applied to any type of missing knowledge, including object identification and categorization, labeling objects with words, learning action models, or labeling actions.
3. *Interactive*: When the agent's knowledge is insufficient to make progress on a task, or the instructions are incomplete or ambiguous, the agent can ask for help. In a mixed control setting, the instructor can ask the agent for information regarding its state and the environment, verify an agent's learning by questioning the agent, and provide corrections.
4. *Knowledge-level interaction*: The instructor provides knowledge to the agent by referring to objects and actions in the world, not the agent's internal data structures and processes.

To explore the potential of learning with situated instructions, we are developing a robotic agent that learns new nouns, adjectives, prepositions, and verbs in a simple real world table top robotic environment. There is a single robot arm that can pick up colored foam blocks, and there

is an overhead Kinect camera, with associated software for segmenting the imaging and extracting color, shape, and size information about the objects. To learn nouns and adjectives, the agent must learn to associate linguistic forms (words such as *red*) to perceptual features (RGB values). The agent acquires prepositions by associated linguistic descriptions (*to the right of*) provided by interactive instruction with a set of spatial primitives.

The new verbs are learned by generalizing from a situated example execution. The instructor leads the agent through an example of a composite action (*move*) which is a composition of *pick-up* and *put-down* actions. The agent uses explanation based generalization (chunking) to learn a general execution of *move.* In order for this type of instruction to work, the agent must have a pre-existing set of primitive actions that are known to the instructor.

Below is a painfully brief overview of the processing steps involved in situated interactive instruction.

1. Task Performance: The agent attempts to perform a task using existing linguistic, procedural, semantic, and spatial knowledge.
2. Detection: The agent detects that its available knowledge is incomplete (and impasse arises).
3. Retrieval: The agent attempts to retrieve prior instructions from episodic or semantic memory that are relevant to the current situation. If successful, goes directly to steps 6 and 7.
4. Acquisition: The agent requests help from the human, who instructs the agent. The agent creates a declarative representation of the instructions in its short-term memory, which is automatically stored in episodic memory.
5. Comprehension: The agent maps the instructions to the current situation, making connections between the words in the instructions and perceived objects and actions. If comprehension fails, possibly because of ambiguous instruction, further instruction is requested.
6. Execution: The agent executes the action, which may involve a change to the agent's internal state (such as associating a label with an object) or performing an action in the world.
7. Retrospective Reflection: Following execution, the agent can reflect back to understand what went right or what went wrong with the instructions. This is an internal debrief that allows the agent to review and further generalize and learn from its experiences.

The agent learns nouns and adjectives by storing associations between the words and perceptual features in semantic memory. Prepositions are acquired by deliberately storing associations between linguistic forms that describe spatial relationships between objects to the spatial primitives extracted by the spatial visual system (and then refining this set through additional experience.)

For learning verbs and associated actions, the agent simulates the instructions on an internal model to verify that it understands why it is performing the instructions, which allows the agent to generalize the instruction to future situations, reducing the need for future instruction. As a side effect of execution, Soar compiles the processing (via chunking) into new rules that avoid future impasses.

Interaction Model

One of the challenges of interactive instruction is that the agent must maintain a representation of the state of interactions with the instructor while acting in the environment, and then learn from the instructions in the context in which they were provided. Thus, the agent needs a model of task-oriented interaction. Such a model is required to support the properties described below.

1. Both the instructor and the agent can assume control of the interactions at any time.
2. The interaction model provides a context for instructor's elicitation, allowing the agent to take relevant actions.
3. The interactions by the agent should be informed by agent's reasoning, learning, and acting mechanisms.
4. The interaction model and the sequence of interactions should inform agent's learning.

The interaction model we use has been adapted from Rich and Sidner (1998). It captures the state of task-oriented interaction between the agent and the instructor. To formalize the state of interaction, we introduce (1) *events* that change the state of interaction; these include dialog utterances, actions, and learning, (2) *segments* that establish a relationship between contiguous events, and (3) a *focus-stack* that represents the current foci of interaction.

In accordance with the discourse interpretation algorithm described by Rich and Sidner (1998), each event changes the focus-stack by, (i) starting a new segment whose purpose contributes to the current purpose (and thus, pushing a new segment with a related purpose on the focus stack), (ii) continuing the current segment by contributing to the current purpose, (iii) completing the current purpose (and thus eventually popping the focus stack) or (iv) starting a new segment whose purpose does not contribute to the current purpose (and thus pushing a new, interrupting segment on the focus-stack, changing the purpose of the interaction). An event contributes to a segment, if (i) it directly achieves the purpose, and (ii) it is a step in achieving the purpose.

Results

Using the approach described above. The robotic agent acquires new adjectives such as *red* and *small* and nouns such as *triangle*, *rectangle*, and *arch* for novel objects. Typically, the agent requires a few interactions and situated examples of objects to acquire these linguistic forms. After learning, the agent can classify novel objects along the dimensions previously observed and can associate correct nouns and adjectives to them. The learning is verified by asking questions such as *"Which is the blue rectangle?"*

To teach novel prepositions and prepositional phrases (such as *to the right of*), the agent is provided with examples of objects that satisfied the desired spatial

relationships along with the phrase via interaction. The learning is verified by arranging objects and querying the agent using questions like "*Which object is to the right of the blue rectangle.*" The agent is the able to provide appropriate answers to such questions.

The agent can also be taught a new verb, such as *move*, by leading it through a situated execution of the composite action. The example execution involves picking up a blue rectangle and placing it at the desired position. Through internal explanation, the agent learns a general execution of *move*, and is able to execute other instances of *move*, but with different arguments, such as *move* a red rectangle.

The learning is embedded within the interaction and action execution framework. The instructor can ask the agent to perform an action, or describe an object. When the agent encounters unknown words, such as "Pick up the red triangle," where either "red" or "triangle" are undefined, the agent engages the instructor in an interactive dialog to learn the unknown words.

Conclusion

The purpose of this paper was to describe how the Soar cognitive architecture supports creating flexible and adaptive cognitive robotic agents. It incorporates both symbolic and non-symbolic processing, has multiple learning mechanisms, and has been applied to multiple robotic systems. Each of the areas we presented is an active research project, and there is much to be done on them individually. In addition, one obvious place for future work is their integration.

A second place for future research is to push further on the interaction between low-level perception and high-level cognition. SVS provides one level of processing where these come together, but as of yet, we do not have a general theory (or implementation) of adaptive low-level perception that translates noisy pixels into object descriptions, categories, features, and spatial relations.

Acknowledgments

The work described here was supported in part by the Defense Advanced Research Projects Agency under contract HR0011-11-C-0142 and by the Office of Navy Research under grant number N00014-08-1-0099. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the DARPA, ONR, or the U.S. Government.

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., Qin, Y. (2004). An Integrated Theory of the Mind. *Psychological Review, 111 (4),* 1036-1060.

Benjamin, P., Lonsdale, D., and Lyons, D. (2006). Embodying a Cognitive Model in a Mobile Robot, *Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision*.

Benjamin, P., Lonsdale, D., and Lyons, D. (2007). A Cognitive Robotics Approach to Comprehending Human Language and Behaviors. *Proceedings of the 2nd ACM/IEEE International Conference on Human-Robot Interaction*, 185-192.

Benjamin, P., Lyons, D., and Lonsdale, D. (2004). Designing a Robot Cognitive Architecture with Concurrency and Active Perception, *Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics*.

Dellaert F. and Kaess, M. (2006). Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing, *International Journal of Robotics Research. 25 (12) 1181-1203.*

Huang, A., Olson, E., and Moore, D. (2010). LCM: Lightweight Communications and Marshalling, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Huffman, S. B., and Laird. J. E. (1995). Flexibly Instructable Agents, *Journal of Artificial Intelligence Research*, 3, 271-324.

Kinkade, K. R. (2012). Cooperative SLAM of Local and Topological Features for Soar Robot Agents, Technical Report of the Center for Cognitive Architecture, University of Michigan.

Kosslyn, S. M., Thompson, W. L., and Ganis, G. (2006). *The case for mental imagery*. New York, New York: Oxford University Press.

Laird, J. E. (2012). *The Soar Cognitive Architecture*, MIT Press.

Laird, J. E., Derbinsky, N., Voigt, J. R. (2011). Performance Evaluation of Declarative Memory Systems in Soar. *Proc. of the 20th Behavior Representation in Modeling and Simulation Conf.* 33-40.

Laird, J. E., Hucka, M., Yager, E., and Tuck, C. (1991). Robo-Soar: An integration of external interaction, planning, and learning, using Soar, *IEEE Robotics and Autonomous Systems*. 8(1-2), 113-129.

Laird, J. E., and Rosenbloom, P. S., (2000). Integrating execution, planning, and learning in Soar for external environments, *Proceedings of the National Conference of Artificial Intelligence*, 1022-1029.

Langley, P., Laird, J. E., and Rogers, S. (2009). Cognitive architectures: Research issues and challenges, *Cognitive Systems Research*, 10(2), 141-160.

Lathrop, S. D., Wintermute, S., Laird, J. E. (2011), Exploring the Functional Advantages of Spatial and Visual Cognition from an Architectural Perspective. *Topics in Cognitive Science*, 3, 796-818.

Marinier, R., Laird, J. E., and Lewis, R. L. (2008). *A Computational Unification of Cognitive Behavior and Emotion.* Journal of Cognitive Systems Research.

Quinlan, J. R. (1992). Learning with Continuous Classes. *Proceedings of 5th Australian Joint Conference on Artificial Intelligence*. Singapore. 343-348.

Rich, C., and Sidner, C. 1998. COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*.

Toussaint, M., and Vijayakumar, S. (2005). Learning Discontinuities with Products-of-Sigmoids for Switching between Local Models. *Proceedings of the 22nd International Conference on Machine Learning*, 904-911.

Wintermute, S. (2009). Integrating Reasoning and Action through Simulation, *Proceedings of the Second Conference on Artificial General Intelligence*.

Wintermute, S. (2010). *Abstraction, Imagery, and Control in Cognitive Architecture*. Ph.D. Thesis, University of Michigan, Ann Arbor.

Wintermute, S. and Laird, J. E. (2008). Bimodal Spatial Reasoning with Continuous Motion, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, Chicago, Illinois.

Xu, J. Z., Laird, J. E. (2010). Instance-Based Online Learning of Deterministic Relational Action Models. *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. Atlanta, GA.

Xu, J. Z., Laird, J. E. (2011). Combining Learned Discrete and Continuous Action Models. *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. San Francisco, CA.