Expanding Task Diversity in Explanation-Based Interactive Task Learning

Aaron Mininger MININGER@UMICH.EDU
Computing Science and Engineering, University of Michigan, Ann Arbor, MI 48103 USA

1. Introduction

Artificially intelligent agents with interaction capabilities are becoming ever more prevalent. As their capabilities increase and they start operating in more complex environments, it will be increasingly difficult to pre-program them with all the behavior desired of them. People will want to direct, customize, and extend their capabilities over time, and do so easily so that it does not require an experienced programmer or thousands of demonstrations to instill a new ability. Fortunately, humans already possess great ability to teach others quickly, and it is the goal of the research problem of Interactive Task Learning (ITL) to leverage this teaching ability to learn new tasks from humans through natural forms of interaction.

Interactive Task Learning can take different forms and there are diverse approaches that address specific aspects. In this dissertation we will focus on Situated Interactive Instruction, where the instructor and the agent are *situated* in a shared environment, both the instructor and agent are engaged in a bidirectional *interaction*, and where the primary form of communication is natural language *instruction*. This is a powerful way of learning, as language is *information-dense* and can contain knowledge specific to the current situation and precise needs of the agent.

However, learning a task through Situated Interactive Instruction presents a number of difficult challenges. Since human instruction is time-consuming and the number of examples are limited, the agent must learn quickly and extract a large amount of useful information from few examples. Building an ITL agent that can learn from instruction requires the integration of a number of diverse capabilities that themselves are difficult, including perception, action, natural language understanding, grounding, and dialog management. In addition, the agent will require a large amount of existing knowledge about the world and itself to understand and learn from instructions. Based on these challenges and constraints, below is a summarized list of characteristics that an ideal agent that can learn new tasks from situated interactive instruction should demonstrate:

1. *Competent Execution:* Once the agent learns a task, it should correctly perform that task in the future without making mistakes.

2. *Rapid Learning:* The agent should learn quickly from few examples, since human instruction is time consuming and bandwidth limited. The agent should be able to immediately use the knowledge it's been given to make progress on the task, and perform the task in the future with minimal additional instruction.

3. *Instruction Efficiency:* The agent should use reasoning and planning when possible to reduce the need for instruction, and should learn as much as possible from each instruction.

4. *Generalizable Knowledge:* The agent should be able to generalize task knowledge to future variations of the task without needing further instruction.

5. *Compositional Knowledge:* The agent should build upon previously learned tasks and concepts and use them to learn even more complex behavior in the future. Tasks learned through instruction should be seamlessly used alongside tasks it was preprogrammed with.

6. *Diverse Learning Abilities:* The agent should be able to handle a wide range of types of tasks and different forms of instruction.

7. *Correctable Knowledge:* The agent should allow the instructor to correct its knowledge when there was a mistake made by the instructor during teaching or by the agent during learning.

8. *Robust Performance:* The agent should be able to detect, handle, and recover from errors that arise during action execution, perception, environmental changes, etc.

Discovering and developing approaches which have all of these characteristics is an enormous challenge, made more difficult by the sheer complexity and diversity of tasks which humans will want to teach. The agent will need to learn tasks with wildly different characteristics, featuring diverse types of objects, concepts, actions, and instructional strategies. However, much of the current research in instructional ITL is limited to simple tasks that do not feature this complexity, with many being variations on pick and place tasks (a more detailed overview is provided in the later related work section).

In this dissertation, we will build on a previous approach to instructional ITL by Huffman and Laird (1995) and later Mohan and Laird (2014) and investigate how to extend this approach beyond a tabletop blocks-world domain to learn a substantially more diverse space of tasks. Mohan's approach was to learn general task representations from the given instructions and then use explanation-based techniques (DeJong and Mooney, 1986; Mitchell, Keller, and Kedar-Cabelli, 1986) to learn procedural knowledge that applied that general knowledge to a specific task instance and execute the task. This approach does well with respect to several of the desired characteristics above. It can successfully perform a task after a single teaching interaction (1, 2) and can learn the task execution policy through internal reasoning and planning without needing to be explicitly taught (3). It can successfully generalize to variations of the task with involving different objects (4) and learn hierarchical tasks (5). However, it is an open question as to whether this approach can scale to more complex and diverse tasks in more complex domains. This dissertation will investigate how this approach can be extended to learn more diverse tasks specifically along three main dimensions of complexity: *diverse actions*, *task formulations*, and *task modifiers*.

Diverse Actions: First, ITL agents will need to learn and execute tasks that involve more than just the physical manipulation of objects. Many common tasks include actions that involve perceptual acts (checking if the lights are on), communicative acts (making an announcement), or mental

acts (recalling that milk is stored in the fridge). Since the agent builds up tasks from a basic set of known actions, these initial actions need to include these kinds of capabilities. However, there is a trade-off between keeping this initial set of actions small to reduce the amount of pre-programmed knowledge needed in the system, but still having enough to enable a wide range of more complex behaviors. We will investigate this trade-off and develop a small initial set of actions that are highly compositional and widely useful. There is also the challenge of representing these actions in a way which is compatible with the planning and explanation-based learning capabilities of the agent. This includes having proper preconditions, action models, and state representations.

Diverse Task Formulations: Second, this approach has only been applied to tasks which are formulated as achieving a declarative goal. However, many tasks are not easily formulated this way. For example, imagine a patrol task where the agent needs to drive to four different locations in a particular order. It would be easier to describe the task as a procedure where it follows a given sequence of actions than trying to describe a final goal-state. Or consider the task of following a person at a given distance. This is less about achieving some desired final state via planning and closer to an ongoing optimization or reinforcement learning problem. Given the large diversity of task characteristics, a single formulation (which includes the specific representations, learning methods, and execution approaches used) is unlikely to be applicable to all tasks.

In this dissertation we will investigate whether our approach, which was only applied to learning goal-based task formulations, can be extended to learn other types of task formulations such as a procedure, optimization task, or RL problem. A major challenge is to learn these other formulations in a way that is integrated and consistent with the existing approach. The expected benefit to a unified approach is that the agent will be able to combine these heterogeneous types of tasks in interesting ways, including vertically (a task of one type with a subtask of another), horizontally (a task with subtasks of different types), and blended (a task with characteristics of multiple formulations).

Diverse Task Modifiers: Third, task commands are often given with additional clauses and terms that specify, modify, and constrain the task (e.g. where, when, or how the task should be done). So while there may be a simplest, canonical version of a task, an agent should be able to learn and perform these variations. This will require greater generalization capabilities and more complex representations and reasoning. In this dissertation, we will investigate whether our approach can extend to learning and performing these task variations, focusing on conditional, repetitious, enumeration, and temporal modifiers. In addition, we will adapt or develop a taxonomy for categorizing ways of modifying tasks in the context of interactive task learning.

A crucial requirement for this work is that these extensions are developed in a way that is integrated, consistent, and unified with the previous approach. This means that they share the same representations, same problem spaces, and methods of learning. For example, adding support for learning procedural formulations should not involve adding an additional module that interfaces with the system, but where the internals are fundamentally different. We believe that this unified approach will result in a number of benefits. The major strengths of this approach are its ability

to learn quickly and to generalize, so we want those same benefits to apply to these more complex tasks. It will allow for diverse types of tasks to be composed or blended together in useful ways. And it will bring more clarity to how our specific approach can extend to these more complex tasks, and what its strengths, shortcomings, and limits are.

In addition to significantly increasing the task learning capabilities of our approach, this dissertation will also include the development of a taxonomy of task complexity that defines the space of learnable tasks. This will provide a way to better compare different approaches and their capabilities along different dimensions of complexity. A contribution of this work will be providing a survey of the current field of Interactive Task Learning and categorizing their capabilities according to this taxonomy. To identify how much coverage our approach has with respect to other work (what fraction of their tasks can our agent also learn), this thesis will include replication studies where possible. This will also serve as one of the major evaluative undertakings of this work.

There are several desired capabilities of an ideal ITL agent in the earlier list that will not be a focus of this work. We will assume that the instructor is an expert, and therefore knows the correct language to use and the correct way to instruct the system. While the instructions will be realistic natural language sentences, there might only be one or two correct (and sometimes esoteric) ways of describing a given concept. We will not investigate how to recover from instructional or learning errors (7), or how to recover from perceptual or action errors (8).

It is worth noting that this research is being done with an ITL agent called Rosie, which has been developed in collaboration with others at the University of Michigan. Shiwali Mohan did much of the initial work on task learning and dialog management. John Laird and Peter Lindes have done work with the natural language comprehension. James Kirk has done work on learning games and puzzles through instruction, and learning many of their underlying concepts. His work on concept learning is potentially useful when learning tasks if the task uses a new term or concept the agent doesn't know. Kirk's work does not involve learning new actions or combining actions, but learning game-specific constraints on simple, known actions (usually some variation on the move or write action), such as in Tower of Hanoi where you cannot move a larger block on a smaller one. His work does not involve learning new tasks outside of a specific game/puzzle structure or learning hierarchical tasks. We will ensure that this dissertation clearly identifies the boundaries of previous work to this work.

In the following sections we will first give an overview of our task learning agent and describe how it learns tasks (section 2). We will then describe different types of ITL and review related approaches (section 3). The following three sections will describe our plans on the three dimensions of task complexity we are focusing on: diverse actions (section 4), formulations (section 5), and task modifiers (section 6). Finally, we will go through some examples of learnable tasks which display these different dimensions (section 7) and describe our planned experiments and evaluation of the system (section 8).

2. Agent Overview

In this section we will give an overview of our approach to Interactive Task Learning. Our approach is inspired by research in cognitive architectures and involves first learning a general declarative

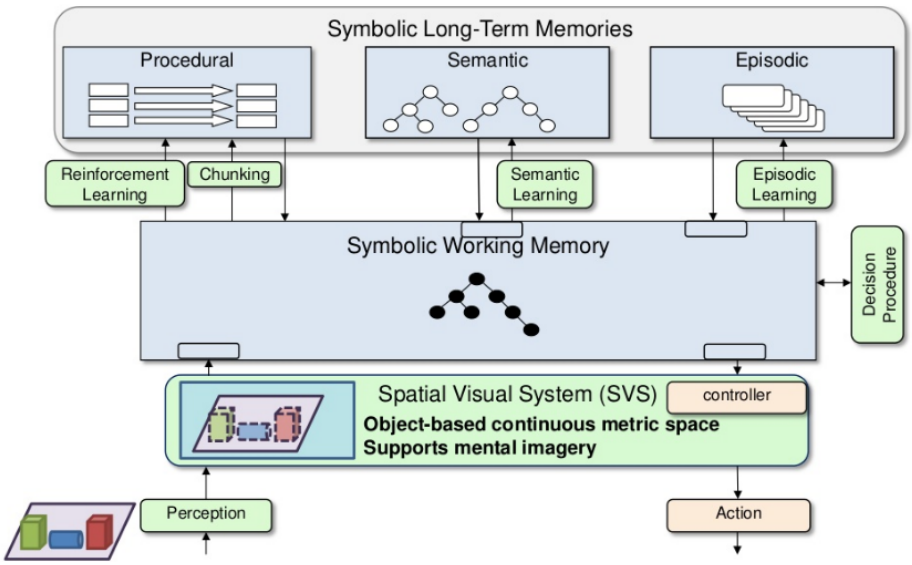
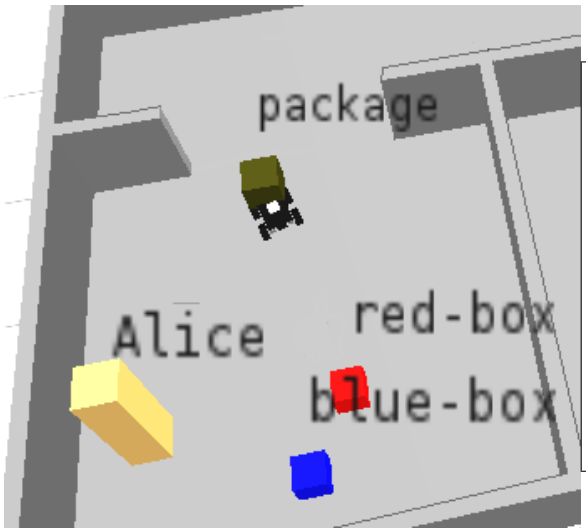


Figure 1: The Soar Cognitive Architecture

representation of the task which is then compiled into procedural knowledge about how to apply that knowledge to the current specific task. It also learns procedural knowledge for how to initiate and perform the task. The task representation is built up by incorporating knowledge obtained from instruction or from the agent's reasoning and problem solving. A fundamental characteristic of this approach is that the agent is learning the exact same representations and types of knowledge that an expert would program if implementing the task by hand. That is, once a task is learned there is no representational difference between it and one that had been pre-programmed.

This agent is implemented in the Soar cognitive architecture (Figure 1). It has a declarative working memory, which contains the agent's current goals, task, beliefs about the world, and dialog state. Working memory contains a declarative description of the immediate environment, which is updated with new information from the perception input buffer and which has links to a spatial, metric representation in the agent's Spatial Visual System (SVS). Processing in Soar is done via procedural knowledge which matches against working memory. In Rosie, procedural memory contains the knowledge about how to parse sentences, maintain the world belief state, handle perceptual updates, learn from instruction, and execute tasks. Soar has a declarative long-term semantic memory which contains knowledge about word meanings (constructions), world concepts, and general task representations. The third long-term memory is a declarative episodic memory, which automatically records a history of the agent's working memory. In Rosie, it is primarily used to retrieve the initial and intermediate world states that existed during the execution of a task. These are used when the agent is learning the task policy.

The first task learning agent implemented in Soar was developed by Huffman and Laird (1995). It learned many of the same kinds of task knowledge in a simulated blocks-world environment, although using different methods. Later, Shiwali Mohan implemented the first version of this specific



package(e_1) obj(e_1)
grabbed(e_1) red(e_2)
box(e_2) obj(e_2)
alice(e_3) person(e_3)
office(e_4) loc(e_4)
blue(e_5) box(e_5)
in(e_2, e_4)
left(e_2, e_3)
near(e_5, e_2)

Figure 2: A simulated world for the mobile robot and some of the corresponding state predicates.

task-learning agent named Rosie (Mohan and Laird, 2014). It was limited to learning simple pick and place tasks, such as *stack* or *store*, in a fully observable environment. Since then, the Rosie agent has been deployed on three platforms, including a tabletop arm, a 4-wheeled mobile robot, and the Fetch robot with a 7 DOF arm, as well as simulated versions of those robots. The same task-learning agent operates across these domains by relying on a uniform representation of the world and basic actions. Each embodiment requires domain-specific procedural knowledge of how to process perception to maintain a stable representation of the agent's current beliefs about the world, and how to perform a set of primitive actions, such as `pick-up(obj)` or `go-to(loc)`.

2.1 Agent Organization

The agent's beliefs about the world are represented in working memory as a set of entities $E = \{e_i\}$ and predicates over those entities $P = \{p_i\}, p_i = \texttt{name}(e_1, e_2, ...)$. Predicates over one entity are called properties, such as `visible(`e_1`)`, `red(`e_2`)`, or `table(`e_3`)`, and predicates over two entities are called relations, such as `in(`e_1`, `e_2`)` or `holding(`e_3`, `e_4`)`. Entities are categorized as either *objects*, *persons*, or *locations*. Some example predicates are shown in Figure 2.

The instructor can interact with the agent using natural language through a chat interface. When a new sentence is received, the agent parses and grounds the sentence. The parser is written in Soar (Laird, 2012) and utilizes an Embodied Construction Grammar (Lindes and Laird, 2017). It produces a semantic representation of the sentence that is connected to concepts in its working and semantic memories, many of which are grounded in its perception of the environment. When the agent communicates with the instructor, it produces a message structure which is converted into English using simple templates.

2.2 Learning Tasks

In order to fully learn a new task, the agent must learn its structure, goal, task decomposition, policy, preconditions, and postconditions. The agent learns a general declarative representation of the task knowledge called the Task Concept Network (TCN), stored in long-term semantic memory. An example TCN can be seen in Figure 3 for the task *'Discard the soda.'* This network contains information about the argument structure of the task, for example, *discard* takes a single entity argument. These argument placeholders are called *slots*, and are used to connect arguments in the task with those in the subtasks and goal. This is the key to the task learning generality, as one slot can be filled in with many different entities, and the TCN shows how that argument connects to the goal and subtasks.

It also learns procedural rules that apply the learned knowledge to the current situation when the agent is actually executing the task. The learning is *impasse driven*, which means that the agent attempts to execute the task until it is missing some knowledge required to make progress. When it detects that it is missing knowledge, it tries to obtain it through internal reasoning or initiating an interaction with the instructor. For example, if it is missing the knowledge of which action to perform next, it will do some internal planning to find the next action, or ask the instructor what it should do. It then interprets the instruction, learns from it, and continues performing the task. In the following sections, we describe how each of these aspects of the task is learned in greater detail with the example task *'Discard the soda'*, which involves putting a soda can into the garbage.

2.2.1 Task Structure:

When the agent receives a task command with a new name, it creates a new TCN and extracts the argument structure from the command, replacing references to specific entities or predicates with *slots*, which are placeholders that will be filled in during a specific task execution. For example, with *discard,* there is a single argument, an entity (the soda), so the TCN is initialized with the task name and a single entity argument slot. More complex commands can have multiple arguments, e.g. *'Deliver the green box to the office.'* Refer to Figure 3 for the full TCN learned for discard.

2.2.2 Goal:

Once the agent has created a task structure, it attempts to execute it, but reaches an impasse because it does not know the goal. It then asks the instructor *'What is the goal?'* and the instructor can reply with a statement such as *'The goal is that the soda is in the garbage'*, which will be interpreted as a set of one or more predicates: {in(e_{soda}, $e_{garbage}$)}. Then the agent adds a representation of the goal to the TCN. The goal contains several different arguments (in this example in, e_{soda}, and $e_{garbage}$), and for each the agent needs to map them onto the task arguments. Here the agent uses the heuristic that if an argument appears in both the task command and the goal then it is mapped to the same slot. For example, e_{soda} appears in both the task command and the goal, so the corresponding arguments in the task and goal representations are mapped to the same slot. This is referred to as an *explicit argument*, since it is explicitly mentioned in the task command (*'Discard the soda'*). This mapping is a key factor in the agent's ability to generalize to other task variations. If the same command is used with a different object (e.g. *'Discard the newspaper'*), the TCN shows how to

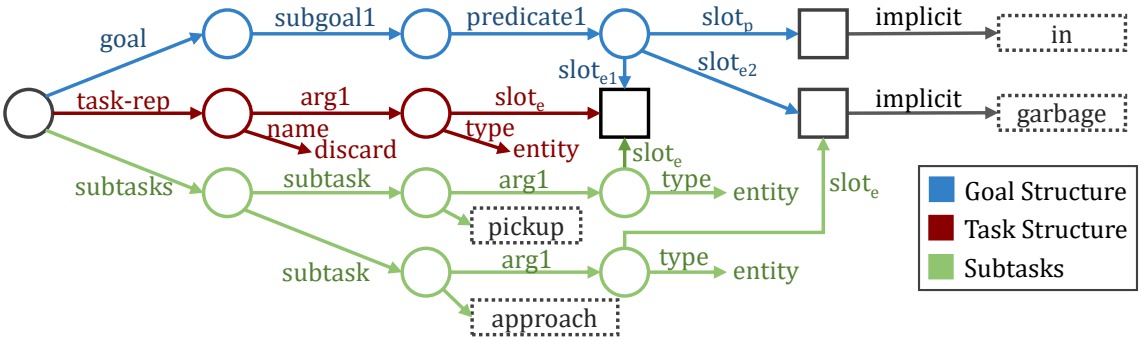


Figure 3: Task Concept Network for the *discard* task. Argument slots are shown as squares. Dashed rectangles represent the root nodes of other concepts in semantic memory. The third subtask for `putdown` is omitted.

map that different object to the goal. If a goal argument is not present in the task command (e.g. `in` and $e_{garbage}$), it is assumed to be fundamental to the goal. It is referred to as an *implicit argument*, and is added as a new slot with implicit values (descriptions of the argument that can be matched against the state). These arguments will not be generalized, i.e. the goal of discard will always reference the garbage.

The agent then constructs a representation of the goal in working memory by matching the mapping information in the TCN against the world and task instance. This goal is connected to specific entities in the world, either matched through the argument mappings in the task structure (for explicit arguments, such as the soda) or through the implicit values for that slot (for implicit arguments, such as the garbage). As a side effect of this processing, a rule is learned through Soar's chunking mechanism that incorporates the mapping logic in its conditions. This rule will create the goal given the current task. In the case of *discard*, it learns a rule summarized as [`if task=discard(`e_i`) and garbage(`e_j`) then goal=in(`e_i`, `e_j`)`]. This is a rule that applies to a discard task even if there is a different (non-soda) argument.

2.2.3 Task Decomposition:

To actually execute the task, the agent must decompose it into a sequence of subtasks. Each subtask is either a previously learned task or a hand-coded task that the agent initially knows how to execute (such as `pick-up` or `go-to`). If it has learned a goal for the task, it searches for a plan that will satisfy the goal. If this search is successful, it will perform the first subtask in the plan. If the planning fails, it asks the instructor *'What should I do next?'* and the instructor can respond with the next subtask (e.g. *'Pick up the soda.'*). The agent performs that subtask in its environment, and the process repeats until the goal is achieved. In this example, it might be able to identify the next two steps on its own (`approach(`$e_{garbage}$`)` and `put(`e_{soda}`, `$e_{garbage}$`)`) thus avoiding further instruction.

The agent learns each subtask through a process that is very similar to learning the goal. First it stores a general structure of the subtask in the TCN, where the arguments are mapped to slots in

the same way as they are for goals (see Figure 3). Then, it matches the TCN against the current task and world to generate a representation of the subtask as a Soar operator proposal. Through chunking, it also learns a rule to propose that subtask during the parent task that is connected to specific entities in the world. This proposal rule includes additional constraints inherent to the subtask, such as putting down an object requires that the object is grabbed. For the discard task, it would learn a rule summarized as [`if task=discard(`e_i`) and visible(`e_i`) and not grabbed(`e_i`) then propose pickup(`e_i`)`].

2.2.4 Policy:

Once the task is successfully completed, the agent knows which subtasks are used in the parent task and has an example sequence of those subtasks, but it does not have a specific policy for *when* to do each subtask. It must learn a policy mapping a state to the appropriate subtask $\Pi : S \rightarrow T$. However, the agent has only one example for each subtask, and needs to determine which aspects of the state were actually important for selecting the subtask. The agent does a retrospective analysis of its experience and uses its domain knowledge to generate an explanation of how those subtasks led to the goal being satisfied. First, it retrieves the initial state (state of the world when the task began) from Soar's episodic memory, along with the sequence of subtasks that it executed. It then repeatedly selects each subtask in order, and internally simulates the effects of each subtask on the state until it achieves the goal. Soar's chunking mechanism identifies the features of the state that caused each subtask to succeed, and learns a rule that prefers the executed subtask in each state. This EBL approach ignores irrelevant aspects of the state and learns general rules that apply to new and different situations. For the discard task, it learns three policy rules, one for each subtask. For example, the learned policy rule for *put-down* can be summarized as: [`if task=discard(`e_i`) and grabbed(`e_i`) and garbage(`e_j`) and near(`e_{self}`, `e_j`) then perform putdown(`e_i`, in(`e_j`))`].

2.2.5 Preconditions:

The knowledge it learns from the above methods is sufficient for the agent to perform the task in the future. However, to truly learn hierarchical tasks, the agent must be able to use and plan with a learned task. Thus it must learn the preconditions (when it can perform a task) and postconditions (what the task accomplishes), so in the future the subtask can be used in other higher-level tasks.

To learn the preconditions, the agent again retrieves the initial state when the task began, and simulates performing the task using the policy. If it succeeds in reaching the goal, it learns a rule to propose the task in a similar manner to learning the policy. The rule will incorporate the features of the state that had to be present in order for the entire task to succeed. For discard, this would be: [`If object(`e_i`) and visible(`e_i`) and garbage(`e_j`) then propose discard(`e_i`)`]. Note that this is a rule that describes the conditions required for discard to be a valid action. It is not a policy rule, since it does not represent when an action should be taken within the context of a parent task.

2.2.6 Postconditions:

The agent does not have sufficient domain knowledge to model all of the side effects of each subtask. For example, when going to the garbage, it cannot model which new objects may become newly visible or which objects are no longer visible. Thus, when the agent learns how the subtask changes the world (postconditions), it only includes what the agent *intends* to accomplish. It uses the learned representation of the goal as the postconditions, thus the postconditions for discard would be to add `in(`e_{arg1}`,` $e_{garbage}$`)` to the state. Other auxiliary changes are possible, such as a change in the robot's location, but these are not included. A benefit of this is that the agent will not use a task like discard as a way to change its position.

2.3 Capabilities and Limitations

The original work by Mohan and Laird (2014) demonstrated that this approach can learn hierarchical tasks through instruction in a blocks-world tabletop environment. However, the approach was only demonstrated on a handful of tasks that mostly involved pick up and put down actions along with a few simulated actions such as opening and closing a door. Some example tasks were *'Place the red block to the left of the green block'*, *'Store the red object'* (in the pantry), and *'Set the table'* (whose goal was a few blocks in a particular spatial arrangement on the table). The goal could contain objects that were not referenced in the task command, and given the goal, the agent could perform a search to perform the task without needing explicit instructions. The agent would also learn a generalized state-based policy that could accommodate variations in the arguments and procedure without needing further instruction. However, the tasks were limited to actions and goals that involved singular arguments (i.e. no collections or multiple objects) and where the agent could simulate achieving the goal through applying STRIPS-style changes to the world predicates (Fikes and Nilsson, 1971). If this simulation failed, the agent would learn nothing about how to perform the task. It also required a fully observable environment with a simple set of objects and predicates.

Since this version of the agent, I have made significant enhancements to the agent's capabilities. I extended the agent's modeling of the environment to allow it to operate and learn tasks in a partially observable environment where the agent had to incorporate knowledge about object visibility into the task reasoning (Mininger and Laird, 2016). These extensions also included the ability to operate and navigate across multiple rooms. This involved extending the set of primitive actions beyond *pick-up* and *put-down* to include navigation, perceptual, communicative, and memory-based actions (detailed in Section 4).

I have also made extensions to the task learning capabilities. These include modifying the Task Concept Network to include declarative representations of the subtasks and sequences of goals. This was done as part of extending the agent to learn procedural tasks, not just goal-based ones, allowing the agent to learn tasks with no clear goal or learn a task even when the explanation-based policy learning fails (Mininger and Laird, 2018). This is described further in Section 5.

3. Related Work

From the early days of the field of AI, researchers have been interested in teaching agents through instruction. In 1972, Winograd developed SHRDLU, a program that could understand natural language commands to manipulate a simulated blocks-world environment and learn simple concepts such as specific block configurations (Winograd, 1972). Other early work includes UNDERSTAND (Simon and Hayes, 1976), which could turn natural language instructions describing variations of the Tower of Hanoi problem into internal problem-space representations. There was also the Instructable Production System (Rychener, 1983), a research project with the goal of learning productions from instruction that was unsuccessful but helped develop ideas used in the creation of Soar. One of the earliest examples of learning hierarchical tasks from instruction was Instructo-Soar (Huffman and Laird, 1995), which could learn a full task problem-space representation in a simulated blocks-world environment.

Although Instructo-Soar is an example of Interactive Task Learning, the term ITL and the focus on it as an AI challenge problem has emerged relatively recently. At a 2017 Ernst Strüngmann Forum in Germany, (Gluck, Laird, and Lupp, 2018) a group of researchers came together to better define and formalize the problem of Interactive Task Learning. A working definition was that ITL is a process by which an agent (A) improves its performance (P) on some task (T) through experience (E), where E consists of a series of sensing, effecting, and communication interactions between A, its world, and crucially other agents in the world (Gluck, Laird, and Lupp, 2018). In this dissertation we focus on instructional ITL, where the experience E includes natural language instructions given to the agent by a human teacher, and the agent starts with no knowledge about the task. Another major area of ITL is learning from demonstration (Argall et al., 2009). The demonstrations can take many forms, for example, through kinesthetic demonstrations (Akgun et al., 2012; Phillips et al., 2016) or through observing a human demonstrate an action (Mollard et al., 2015). Learning from demonstration is very useful for learning fine-grained manipulation actions such as opening a drawer. These types of actions are not currently learnable by our agent, and would be difficult to describe only using language. However, it is a difficult problem to extract the relevant features from a high-dimensional and noisy demonstrations, and as a result these approaches tend not to generalize as well from a single example.

There are other areas of research which are related to instructional ITL but which do not cover all of the important aspects, for example, learning a mapping of natural language commands onto actions. Sometimes this mapping is onto actions that the agent already knows how to do (Chen and Mooney, 2011; Matuszek et al., 2013; Lignos et al., 2015). Other research involves learning both the parsing and the actions themselves to generate motion paths from natural language commands (Tellex et al., 2011; Howard, Tellex, and Roy, 2014). Some have even learned how to map higher-level task commands onto actions (Arumugam et al., 2017) or goal predicate representations (Misra et al., 2015). These approaches use natural language, but the learning is mostly done in batch, not through interaction, and requires significant training data. This supervised data typically consists of both the instructions and the correct actions or plans, which is time consuming to generate.

There have been a number of different approaches to learning new tasks from interactive instruction, which are most closely related to this dissertation. Some have focused on teaching tasks to an electronic personal assistant. Allen et al. (2007) implemented a browser-based agent which could

learn tasks related to searching and summarizing information extracted from web pages. It could learn hierarchical tasks which could include iterative loops. More recently, Petit and Demiris (2016) developed an agent called LIA which could learn tasks in an email domain from a step-by-step set of instructions. It had some generalization capabilities, but could not learn hierarchical tasks. Both of these approaches learned tasks as fixed procedures (i.e. no planning capabilities).

Overall there are two major formulations used when learning tasks from instruction: representing the task as a procedure and representing the task as a goal. For example, one approach learns mobile robot tasks (such as fetching an object) by learning an Instruction Graph that encodes the actions and the control flow between them (Rybski et al., 2008; Meriçli et al., 2013). This graph can include complex control flow such as loops, conditional branches, and termination conditions. More recent work by Gemignani, Bastianelli, and Nardi (2015) is very similar and also learns a procedural graph structure for mobile robot tasks that can include loops and conditionals. Both of these have basic communicative and perceptual actions (perceiving an object and saying a predefined sentence) in addition to navigation instructions. Another example of a task learning agent that learns tasks as procedures is by Frasca et al. (2018), and is implemented in the DIARC cognitive architecture. It can learn about objects and actions through one-shot instruction, where new actions can be learned as scripts which can include some basic perceptual steps. Lallée et al. (2010) describe an agent which can learn to collaboratively assemble a table, however the scope of its task learning abilities are limited to simple named macros. Mohseni-Kabir et al. (2018) show their agent is able to learn a hierarchical task involving rotating tires on a car and can take advantage of background knowledge of a parts hierarchy. While the task representation is a fairly simple version of an HTN, they are able to simultaneously learn new action primitives involving motion planning.

Other approaches learn the goal of a task. When combined with planning, this can allow the agent to be more flexible and adapt to environmental variations. For example, She et al. (2014) compare the initial and final states to extract a goal set of predicates. This can then be used by a discrete planner to execute the task in the future. Later work extended this goal representation to a hypothesis space which could represent multiple possibilities (She and Chai, 2017) The agent can also ask additional questions to prune the space. Suddrey et al. (2017) extract a task representation from an example execution sequence that can be used in an HTN. They further combine this more fixed structure with a planner, and show how it can adapt to certain task variations.

Most of the above approaches do not involve learning any actions beyond physical manipulation or navigation. A few do have some simple uses for having an action be based on perception or saying a specific sentence (Meriçli et al., 2013; Gemignani, Bastianelli, and Nardi, 2015). Others do support learning tasks that involve more than simply moving objects around, such as using a microwave (She and Chai, 2017). None of these are able to learn tasks with all four of the action types identified in the next section. In addition, these approaches only learn tasks using one type of formulation, e.g. a procedural graph, HTN, or goal. These different formulations have their own strengths and weaknesses, and a truly general task learning agent should be able to learn tasks in different ways. Suddrey et al. (2017) does show some ability to combine planning with a HTN representation, producing additional flexibility over a purely procedural representation. However, it cannot learn a task from a goal description alone. Finally, several of the procedural approaches are able to encode task modifiers with control structures such as repetitions, conditional statements,

enumerations, and temporal modifiers. However, there are not clear examples of learning goal-based tasks that can include these types of modifiers.

4. Diverse Action Types

In order for a task learning agent to learn and perform a wide range of complex tasks, it must perform actions that go beyond merely physically performing some action in the world. Consider the task of asking someone a question on behalf of the instructor and then reporting the answer. The actions involved in performing this task include navigating to where that person is,

speaking to that person, remembering their response, and reporting back to the task giver. These actions are not limited to physical actions, but involve perceptual, communicative, and memory-based actions. Thus, in order to learn such a task, the agent must plan with and perform these different types of actions.

This dissertation will explore what makes a useful set of initial action primitives that can be used to build up more complex tasks and which incorporate these different action types. This set should be comprised of action primitives that are useful in a wide range of tasks and contexts and can be effectively modeled during planning and learning. These action primitives should be general so that a minimal initial set can cover the widest range of uses. It is not sufficient to merely identify and implement these actions by themselves, as they need to be integrated into the larger task learning framework. Thus the requirement is not just that the agent be proficient in performing the actions, but that it can also use them during planning, learning, and task decomposition.

4.1 Implementing Different Action Types

Actions in Rosie are similar to STRIPS actions, so in order to add a new action one must define the action's representation and argument structure, preconditions and postconditions, and how to execute the action. The challenge is to develop a representation for each action that can be used effectively during task learning, especially during the retrospective learning where the agent is attempting to generate an explanation of why its actions led to the goal being achieved. For this to succeed, it must have sufficient models of its actions and how they change the world representation. Adding a new action can also involve changing or extending the set of predicates that are in the world to better suit these types of actions. For example, we have added a predicate for whether an object is *visible* to allow reasoning about perceptual actions. Below we describe the four major categories of actions that we have identified. There may be other categories that are worth exploring, and which will be part of this research. From a cursory look at the 25 most common English verbs, their main uses fall into one of these categories (with the exception of verbs that are more descriptive than action oriented, such as *to be* or *to have*).

4.1.1 Physical Actions

Physical actions are those whose primary purpose is to change the physical state of the world and/or the agent. Clearly the specific set of these that the agent starts with is highly dependent upon the environment that the agent is in and the physical embodiment of the agent. In this work the agent is

typically embodied as a mobile robot in a simulated multi-room world that can interact with other objects. We assume that the agent comes pre-programmed with an initial set of simple actions that it knows how to perform and has models of how they affect the world. These are the basic building blocks that the agent can compose to learn and perform more complex tasks. Actually learning these lower-level actions is beyond the scope of this work, and is likely to require other methods than pure instruction. There is a large body of work in learning from demonstration that attempts to learn these kinds of actions and the corresponding motor movements (Argall et al., 2009).

Many of the physical actions that we will need have already been implemented. The initial tabletop arm had the basic primitives `pick-up(obj)` and `put-down(obj, loc)`. For the mobile robot which can drive, we added the actions `go-to(loc)` and `approach(obj)`. We also added many driving actions that can be used in commands, such as `turn`, `drive-forward`, `follow-wall`, and `orient` (to cardinal direction). However, the agent cannot model how a driving action such as turn right will change perception and the world state, so these are not used during planning or explanation. In our simulated domains, we also have a number of actions that the agent can use to interact with more complex objects, such as `open`, `close`, `turn-on`, `turn-off`, and `use(obj1, obj2)`.

Since our approach is limited to learning tasks which are compositions of an initial set of primitive actions, these actions determine the granularity of tasks the agent can learn. For example, if the arm actions are limited to pick up and put down, it could not learn a task that involved sliding or tipping over an object. However, if the agent did have some means of learning these actions (such as learning from demonstration), it could use those to learn new higher-level tasks. This would be an interesting future research problem. It also cannot learn novel uses of a primitive action, or model secondary changes or effects that might be useful.

4.1.2 Perceptual Actions

If the agent is in a partially observable environment, it will sometimes need to perform an action to change its perspective or to perceive something not currently visible. This could involve turning to face an object, driving closer to a tabletop to better view the scene, or searching for a person in a building. We will focus on vision, although this could apply to other forms of sensing, such as finding the source of a noise. Having an agent that can learn and execute tasks in a partially observable environment and where perceptual acts are incorporated into the task hierarchy introduces a number of challenges. The instructor could refer to an object which the agent does not see and does not know where it is. While executing a task, the agent may sometimes need to do additional steps to accomplish some perceptual goal. For an ITL agent such as ours, it may be impossible to fully model how an action (such as driving to another room) will change the perceptual state, making it difficult to plan and generate explanations.

In previous work, I have described extending our agent to be able to learn and perform tasks in a partially observable environment consisting of multiple rooms in an office setting (Mininger and Laird, 2016). This involved changing the world model so that not-visible objects are handled properly, extending the language comprehension to handle references to unseen objects, and learning different strategies for finding objects that were not visible. These changes were crucial for allowing the agent to operate in a multi-room environment. Even a task as simple as delivering a package

to a room would have been impossible for the agent without these changes, as the destination room wouldn't normally be in the agent's working memory.

One of the most significant additions to the task learning was adding a `find` action, whose goal was for an object to become visible. This allowed the agent to incorporate reasoning about object visibility into its planning and explanation capabilities. Since many of the preconditions for primitive actions such as `pick-up` required that the object be visible, the agent would have to perform a find action before performing one of those actions. However, the agent can have different types and amounts of knowledge about an object it is trying to find. Thus inside the find action there were a number of different subtasks it could use, depending on what it knew about the object. `Face(obj)` was used if the agent knew the exact position of an object that was out of sight (e.g. behind the agent) and would turn towards that position. The agent could try and `recall` information about the object from long term memory, such as the last location it saw the object. The `scan` action involved turning 360 degrees to look around the current room to try and spot the object. The `explore` action involved driving to every known location and scanning each one. The agent could also `ask` the instructor for help locating the object.

Having a partially observable environment makes the explanation-based learning much more challenging, because it is extremely difficult to model how the state changes as the result of the agent's movement. The state may or may not change due to properties of the world not represented symbolically. For example, suppose a task involves two objects that are not visible at the start. If the first action `find(obj1)` results in both objects becoming visible, then the agent cannot generate an explanation for why the second became visible (it is based on the physical state of the world, not as a predicted consequence of an action the agent took). There is more work to make the retrospective learning more robust to these types of failures.

4.1.3 Communicative Actions

In some tasks, it is useful for the agent to communicate with others as part of the task, not just as aid in the learning phase. Communication can allow the agent to provide information or to acquire information from other people, for example, delivering a message, asking a question, or making an announcement. This can also increase the flexibility of tasks by allowing them to be customized or directed by other people. For example, the agent could learn a task where it asks someone a question and then does something based on the response, such as asking what drink they would like and getting it for them.

In a previous paper (Mininger and Laird, 2016), we added two basic communicative actions: `say` and `ask`, and learned tasks that involved these actions. Adding these actions required significant extensions to the state representation. We allowed entities to also represent abstract messages, not just physical objects, and added predicates such as `heard` and `answer`. We do not focus on language generation, so any language produced by the agent is given directly by the instructor. Thus a response is either a quoted sentence that the agent should repeat verbatim, or a reference to a specific object (e.g. *'What would you like to drink?'*, *'A soda.'*).

4.1.4 Memory-Based Actions

These actions involve operations with the agent's knowledge, as sometimes a task will require accessing or modifying its knowledge, whether working memory or long-term memory. Some examples include recalling the last place the agent saw a person, recording where a person wanted a package delivered, or retrieving the location of the copier from long-term memory. Allowing the instructor to include these actions opens up tasks to much more useful versions, as the agent can base actions off of more than just the current perceptual state. For example, when learning the task *'Fetch a stapler'*, once the agent finds a stapler it must return to where it started. However, once the agent leaves the starting location that information is not in the normal world representation. The agent learns to record the starting location in its working memory, so it can return there after finding the stapler. We have implemented one action that modifies working memory (remember), and one action that retrieves information from long-term memory (recall).

The `remember` action involves adding predicates to the world state in working memory. This allows the agent to record information that may be needed later. In the *fetch* example from above, the agent must remember the location where it started, but once the agent travels somewhere else there are no predicates that distinguish that starting location. The instructor first gives the command *'Remember the current location as the starting location'*, which causes the agent to add the `starting(`e_{loc}`)` predicate. Later, the agent can return to the starting location and achieve the goal.

Another use for this action is to allow the instructor to use a non-perceptual predicate and teach the agent how to add it. In the task of *'Serve Alex a drink'*, the goal is that *'Alex has a desired drink'*. However, the agent does not know how to identify the `desired` predicate. The agent will ask Alex *'What drink would you like?'* and she will respond *'A soda'*. The agent needs to connect the soda object to the 'desired drink' but does not know how. The instructor gives the command *'Remember the answer as the desired drink'*, which causes the agent to add the `desired(`e_{soda}`)` predicate to the state and gives it enough knowledge to plan how to achieve the goal.

The `recall` action involves retrieving knowledge from long-term memory using information in the command to construct a cue and copying the result into the current state representation in working memory. There are two variants that determine which long-term memory is accessed. Recall by itself cues semantic memory. Given the command *'Recall the office of Alice'*, the agent will construct a cue looking for an identifier in semantic memory that represents Alice and which has an office. If such an office exists, the office is added to working memory. The combination of recall+when indicates episodic memory. Given the command *'Recall the current location when Alice was visible'*, the agent will construct a cue looking for an episode when working memory contained the predicates `Alice(`e_i`)` and `visible(`e_i`)` and copy the current location from that episode into working memory.

These actions have been implemented and tested, but only in some limited cases. The *remember* action has only been tested in the cases described above: remembering the starting location and remembering the answer to a question. The *recall* action has been used when looking for an object to recall an object's storage location, a person's office, or the last location an object was seen. There is more work to make these more general and to explore other ways in which they can be used. Adding these complicates the retrospective learning, because in order to generate an explanation of

why one of these memory actions helped lead to the goal, the agent needs information that isn't in working memory. More work will have to be done in extending the retrospective learning to better include these actions.

4.2 Greater Action Complexity

The actions that have been implemented for each of these categories above represent relatively simple instances. More complex tasks with larger scopes will require even more complex actions that combine in different ways. For example, a task such as cleaning a countertop would involve fine motor skills integrated with enhanced perceptual capabilities. A task as complex as negotiating a meeting time among multiple people will require additional communicative and mental abilities that are not covered by the examples above. Tasks can involve internal reasoning or mental computation, not just accessing and moving knowledge around the agent's memories.

The main focus of this thesis will be demonstrating how a task learning agent can learn tasks that incorporate these types of actions into its reasoning, planning, and learning capabilities. This involves implementing a set of actions that the agent starts with and showing how they are useful across a wide range of tasks. A set of these types of actions consisting of those described above has already been implemented and successfully used during task learning. However, there is still work to be done to quantify how broadly useful these actions are (how many tasks they enable learning), and to explore whether there are any other actions in these categories that would be sensible to add. The goal is for this set of tasks to cover a basic set of use cases across these different categories, not to be a sufficient set of actions to support any task.

5. Diverse Task Formulations

Just as there are different types of actions that the agent should be able to perform, there are also different types of tasks that an ITL agent should be able to learn. Since there is a large variation in task characteristics, the most suitable instructions, representations, learning methods, and execution methods will depend on the characteristics of the specific task. A single approach will not be the best fit for every type of task, therefore a general agent should be able to formulate a task in different ways. A task formulation includes which learning methods are used, what knowledge is learned and how it is structured, and how the agent uses this knowledge to perform the task in the future. Tasks may be formulated in different ways, for example, achieving a goal, following a procedure, or maximizing an objective function. These different formulations each have their own trade-offs in generality, ease of instruction, and ease of learning. In addition, a task might combine elements of these different formulations, for example, a task might involve driving to a destination (a goal) as quickly as possible (maximization). In order for task learning to be truly compositional, these different formulations must be implemented in a unified manner so that a single task can have elements of multiple formulations, and a task formulated in one way can have subtasks that are formulated in different ways.

These different formulations can come from the instructor, where the manner in which the instructor teaches the task can suggest the appropriate formulation. For one task the instructor may prefer to describe the goal of the task and have the agent figure out how to achieve it, for another task

the instructor may find it difficult to describe the goal and so instead provide a specific procedure for the agent to follow. Thus providing the ability to learn different formulations for a task gives the instructor greater flexibility in choosing how to teach the task. In addition, the capabilities and knowledge of the agent can also have an impact on which formulation the agent chooses to use to represent a task. For example, if the instructor did describe the goal of a task, but the agent lacked sufficient knowledge and planning capabilities to figure out how to achieve the goal, it could instead decide to learn the task as a procedure.

Below we introduce a list of different formulations and describe their characteristics. This is an initial list, but more work will need to be done to produce a more formal taxonomy of these formulations, and to determine others that are not included below. A key research question addressed in this dissertation is whether our approach to learning tasks, which has so far focused on goal-based formulations, can also accommodate these other types in a unified way.

5.1 Goal-Based Task Formulation

In a goal-based task formulation, the agent has a declarative representation of the task's goal. The goal could be explicitly taught by the instructor, or learned by the agent through experience. Like the actions in the previous section, goals should not be limited to achieving some physical state in the world, but can include communicative, informational, and perceptual goals. Once the agent has learned the goal, it can accomplish the task by performing actions to try to achieve that goal. The agent could use planning to select the next action or some other method such as reinforcement learning to learn a policy. Once the agent achieves the goal, it is finished with the task.

This was the formulation that was first implemented by Mohan and Laird (2014) which is described in section 2. In it, the instructor provides a natural language description of the goal (such as *'The goal is that the can is in the garbage bin.'*) which is transformed into a set of desired state predicates: `{in(can, garbage)}`. The agent stores a general declarative representation in semantic memory, and learns a procedural rule that elaborates the goal for an instance of the task. The agent uses planning to search for a next action that will help achieve the goal, or if necessary, gets additional instructions from the instructor. Once finished, it learns a state-based policy through EBL by simulating the actions beginning at the initial state and generating an explanation for how they help achieve the goal. For this to work, the agent must have sufficient action models that can simulate the effects of actions and model how the actions achieved the goal. Since this approach relies on using explanation to generate a general policy, a crucial requirement of this dissertation is that any changes and extensions to our approach are compatible with this method of learning task policies.

5.1.1 Negative Goal

Instead of formulating a task as achieving a desired state, the agent could formulate the task as avoiding some undesired state or states. This state is still represented, except now the agent needs to learn a policy that avoids that state. For example, in the task *'Clear the table'*, the instructor could say *'The goal is that no objects are on the table'*. One ramification is that the agent no longer has a clear termination condition; given a negative goal alone does not tell the agent when the task is

finished. One common use would be to use a negative goal as a constraint on a task, for example, *'Bring me a pen, but do not go in the conference room.'* A shortcoming of our approach is that we don't model how the world changes apart from the agent's actions, so we cannot easily stop something from happening unless as a direct, predictable result from the agent's behavior. We plan on allowing negative constraints to be added to normal tasks, but do not plan on addressing this issue of stopping something from happening.

5.1.2 Maintenance Goal

A maintenance goal is where the agent is expected to achieve and/or maintain some condition over time. The main difference is that achieving the goal is not used as a termination condition, so there is no longer a well defined stopping point. An example would be keeping a table stocked with drinks during a marathon, where the goal is to make sure there are at least ten water bottles on the table. This is very similar to the normal goal case, except that it may have multiple learning sessions if the goal is achieved multiple times.

5.2 Procedural Task Formulation

In a procedural task formulation, the agent has a declarative representation of the specific actions the agent needs to take, and the control flow among them, in order to perform the task. It could be as simple as a linear sequence of actions, or it could include complex control flow such as loops, conditions, and enumerations. Once this representation is learned, the agent can simply follow the procedure to execute the task in the future, and when the procedure is finished, the task is complete. Often this procedure is explicitly described by the instructor, and can be easy to teach as the instructor just has to provide a sequence of commands. This is also easier for the agent because it can rely on the procedure instead of needing to learn a policy or perform complex planning requiring rich domain knowledge. For example, one does not need to be an expert in the science of cooking in order to follow a recipe. However, if the procedure does not include how to deal with problems that arise during execution, it can be hard to recover from errors or unforeseen issues.

We have already done some work in learning procedural tasks that involve a linear sequence of actions (Mininger and Laird, 2018). To do this, we expanded the goal representation to allow a sequence of subgoals, where each subgoal could either be a set of state predicates (as before) or the goal of performing a specific action. If no goal is provided and the instructor provides a sequence of commands, the agent records those as a series of goals and then can perform the same actions in the same order later. Because we represented procedural tasks in a way that was consistent with the previous approach, we demonstrated some interesting cases where tasks had a blend of both goal-based and procedural aspects. For example, we showed that the agent was still able to learn the deliver task (e.g. *'Deliver the package to David'*), while missing an action model for `pick-up` (that when it does `pick-up` the object becomes `grabbed`). The agent still had a goal for deliver, but during retrospective learning it could not explain why the `pick-up` action was useful for achieving the goal. However, where as previously it would have failed to learn anything at all, it now learned the first `pick-up` action as a procedural subgoal, and the rest of the actions as a policy. We plan

to expand the ability to learn procedural tasks to include learning more complex control structures such as loops, conditional instructions, and enumerations with sets of objects.

5.2.1 Compositional Task Formulation

A compositional task is a procedural task without the ordering constraints. It is essentially a grouping of subtasks into a single task, where the agent needs to perform the subtasks but in no particular order – like a checklist. When the agent finishes one subtask, it can choose the next from the set of unfinished ones. When all the subtasks are complete, the overall task is finished. There could also be a partial ordering with the subtasks, from instructor preference, environmental constraints, or if one subtask is required before another can be performed. This will require some modification to the goal structure so all subgoals are present at once and the agent can choose to pursue any of them when deciding on the next action.

5.3 Optimization Task Formulation

In this formulation, the agent has a representation of a metric that it tries to maximize (or minimize). This can be provided by the instructor, and the agent can perform actions to try to optimize the state according to the metric. This could be accomplished through planning or reinforcement learning. Unlike a goal-based task, here there is no clear way to know when the agent has achieved the optimal state unless it can somehow detect that it has reached the global maximum. This formulation is also subject to many of the pitfalls that optimization problems can have with large/infinite state spaces that are non-convex.

One common version is the standard interactive reinforcement learning problem, where the agent is trying to maximize a metric (specifically maximizing reward), but there the metric is not explicitly provided. Instead, the agent gets a reward signal from the instructor. This precludes internally simulated exploration, since the agent cannot calculate the reward on its own. But through acting in the world, the agent can learn a policy through RL. This may be a particularly difficult way to learn, since exploring in the real world is slow, instructor time is limited, and realistic state spaces are likely to be huge. This is why having some representation of the reward metric or being able to learn it is very useful.

Our goal is for our agent to be able to learn a reinforcement learning problem space from instruction. This may include learning a representation of the reward function, which features of the state to pay attention to, and which actions to consider. At any one time there will be many of such features and actions which aren't relevant, and including those will greatly increase the time to learn a good policy. Given the discrete state and action representations, it is likely that learning an RL formulation will be limited to simple, proof-of-concept tasks.

6. Diverse Task Modifiers

When an instructor gives a task command to an agent, often it is not in its simplest form but contains phrases that modify how the task should be performed. If a task learning agent can handle these additional constraints and modifiers, it greatly increases the ways in which a task can be used. For

example, consider the simple command of 'Move a cup onto the table'. If this is the only form in which the move task can be used, it limits the usefulness of this task, especially when it is used inside other more complex tasks. There are many different types of phrases that can be used to modify this command to increase the ways in which it can be used. For example, there are temporal clauses ('After dinner is finished, move a cup onto the table.'), conditional clauses ('If the table is clear, move a cup onto the table'), and compound arguments ('Move 3 cups onto the table').

In this dissertation, we will develop a taxonomy of different ways that an action can be modified by adapting ideas from computational linguistics and using what it most relevant to Interactive Task Learning. This can serve as a reference when comparing different approaches to instructional ITL and evaluating what variations of task commands they can learn. Below are some types of task modifiers that we have identified:

- Temporal - when a task should be performed (beginning, end, or duration)
- Conditional - whether a task should be performed (under what conditions)
- Repetitious - how many times a task should be performed
- Spatial - where a task should be performed
- Manner - how a task should be performed (e.g. slowly, quietly, carefully)

We will also work on adding the ability to learn tasks with these different kinds of modifiers, focusing on the first three: temporal, conditional, and repetitious. Previous work did implement some simple spatial clauses, such as *'on the stove'* or *'near the blue block'*, but this dissertation will not attempt to learn more complex spatial descriptions. The last type, involving adverbs that modify the manner in which a task is to be performed (such as quickly, carefully, quietly), would involve more top-down input into how basic motor actions are performed, which is not a current focus of this research (we assume a fixed set of initial action primitives). Previous work could handle only one temporal clause (until) and had no support for conditional or repetitious constructs. In this work, we extend our representations and learning mechanisms to handle these different modifiers and use them during task learning and execution. It is important that this be done in an integrated way, so that our agent can learn just as effectively when these modifiers are used and can transfer from one variant of the task to another without needing additional interaction. In the example above of moving a cup, once the agent has learned the basic version of the task, it should be able to complete the other variations with no additional instruction. Below we discuss each of these three types of task modifiers and the proposed work for each.

6.1 Temporal Modifiers

There are three main ways that a temporal clause can modify a task: it can specify the beginning (e.g. after the meeting, when the microwave is off), end (until the cup is full), or duration (for ten minutes) of a task. Adding support for these clauses will involve extending the task representation to include this temporal information and adding additional procedural rules when executing the task to control when it begins and ends. These clauses can also contain information about how the world

changes, which is useful when learning the policy by generating an explanation. For example, if you give the command *'Press the down button until the projector screen is lowered'*, the until clause tells the agent that when the action is finished, the screen will be lowered. So even if it was missing the knowledge about what the button did, it could reason that the goal of lowering the projector screen was achieved because of that action.

6.2 Conditional Modifiers

The instructor may want to directly communicate the preconditions for an action, especially if the agent is unable to learn them on its own. For example, the instructor could say *'If you are fetching a drink, go to the kitchen'* or *'If the lights are off then turn on the switch'*. Since Soar already represents procedural knowledge using if-then rules, handling conditional actions essentially involves translating the action command to a rule. This process is similar to what happens for normal actions, where the agent learns a general rule for proposing that action, except it includes additional conditions.

6.3 Repetitious Modifiers

The instructor may also want the agent to repeat a task multiple times, and could indicate this in multiple ways. For example, suppose the instructor wanted the agent to put four sodas on the table. She could indicate this by giving a repetition count (*'Move a soda onto the table four times'*), a do-until loop (*'Move sodas onto the table until there are four sodas on the table'*), or by adding a count to the referring expression (*'Move four sodas onto the table'*). Sometimes loops may be specified using quantifiers instead, such as *'Store all objects on the table'*. Incorporating these different looping structures should allow the agent to perform these different variations after learning the base version without needing further instruction.

Executing these tasks multiple times should be straightforward if each iteration is independent. However, the difficulty comes in incorporating repetitious tasks as parts of other tasks. These will necessarily require the agent to reason about sets of objects, not just singular arguments as in previous work. Adding this reasoning will substantially increase the complexity of the explanation-based policy learning. There are also complications between dealing with sets of objects and partial observability, as the agent may not be able to see all the objects involved at the same time.

7. Task Examples

In this section, I go through several task examples and categorize their complexity along the three major dimensions described above. You can see a summary of these dimensions in Figure 4. For each of the different dimensions, there are examples of where different agents would fall on a range from having low capabilities along that dimension to having high capabilities. These are not meant to be definitions, but examples of where a certain agent would fall on the spectrum. In these examples, the regular sentences are those spoken by the instructor, and those in italics describe processing or action execution and are not dialog. The agent's responses are omitted for space.

Diverse Action Types (AT)				
1 (Low)	2	3 (Med)	4	5 (High)
Learnable tasks involve simple physical actions in a fully observable world	Learnable tasks include more complex physical actions in a partially observable environment	Learnable tasks also include basic communication and mental operations	Learnable tasks involve complex versions of all types of actions, but they are still still black-boxes	Learnable tasks include all types of actions used and combined in sophisticated, fine-grained ways
Diverse Task Formulations (TF)				
1 (Low)	2	3 (Med)	4	5 (High)
Only learns one type of task formulation for all tasks	Can learn more than 1 formulation, but using distinct mechanisms or components that can't be combined	Can compose different formulations in the same task hierarchy, but each individual task is strictly one type	Can learn tasks which have aspects of different formulations blended together in limited ways	Can learn a wide range of formulations with wildly different characteristics, and use them together in the same hierarchy and even in the same task
Diverse Task Modifiers (TM)				
1 (Low)	2	3 (Med)	4	5 (High)
Learns 1 canonical version of a task and is limited to that version	Can learn different variations of a task with different modifiers, but cannot transfer from one to another	Can learn a canonical version of a task, then carry out commands that contain different modifiers, but can't incorporate into a task hierarchy	Can learn tasks which use the different modifiers as part of their subtasks, but needs to be explicitly taught how	Can transfer learned task knowledge to very different versions of a task, and fully use these variations during planning and task decomposition

Figure 4: Example agent characteristics along each dimension of task diversity, with agents that demonstrate low capabilities to high capabilities.

7.1 Stack

This example involves an agent in a tabletop, blocks-world environment with an arm that can manipulate blocks. It learns to stack one block on top of another after being given a description of the goal. The teaching dialog may go as follows:

> Stack the red block on the blue block.
> The goal is that the red block is on top of the blue block.
> *The agent does some planning and finds an execution sequence to satisfy the goal.*
> *The agent picks up the block*
> *The agent puts the red block on the blue block*

This agent would have low action diversity (AT1), since it is only learning pick and place tasks in a fully observable environment. It is only learning tasks as a goal-based formulation (TF1), and in this case it is not demonstrating any task modifiers (TM1).

7.2 Deliver

This example involves a mobile agent that can drive around multiple rooms and manipulate objects. It can learn to deliver an object and asks the person who to deliver it to. The teaching dialog goes as follows:

> Deliver the letter.
> Pick up the letter.
> Ask 'Who is this for?'.
> Find the answered person.
> Say 'This is for you'.
> Give the letter to the answered person.
> You are done.

This task involves a partially observable environment where the agent does a find action and involves simple communication, demonstrating medium action diversity (AT3). It is also learning the deliver task as a procedure (no goal is given), but has a goal-based subtask (find), demonstrating medium task formulation diversity (TF3). In this example there is still no demonstration of handling diverse task modifiers (TM1).

7.3 Prepare a Conference Room

This example also involves a mobile agent that can drive around multiple rooms and interact with the environment. In this task, it learns to prepare a conference room for a meeting, which includes a number of steps and utilization of previously learned tasks. The teaching dialog goes as follows:

> Prepare the conference room for a meeting.
> Ask 'What time is the meeting?'
> Remember the answer as the starting time.

Open the door in the conference room.
Turn on the lights in the conference room.
If the projector screen is up, lower the projector screen.
Deliver 10 bottles of water to the conference room table.
After the starting time, close the door in the conference room.
You are done.

Throughout this task the agent uses a number of different types of actions, including the mental action of remembering a time. However, the individual actions are still fairly simple, so this is a moderate level of action diversity (AT3). This agent does learn a number of different formulations, as the task as a whole is a procedural task (do these things in order), but has goal-based subtasks in the same hierarchy. In addition, the subtask of lowering the projector screen can involve a blended formulation, since it has a goal (the screen is lowered), but was also learned as a procedure since it did not know what the button did and therefore could not generate an explanation of why pressing the button caused the goal to be satisfied. This blending demonstrates significant capabilities along the diverse task formulations dimension (TF4). This task also includes a couple task modifiers, including delivering 10 bottles, having a conditional statement for the projector screen, and waiting until a certain time before closing the door. These are not only understood but incorporated into the learned task hierarchy (TM4).

7.4 Higher Level Tasks

The work in this dissertation will include the ability to learn all three of those tasks listed above. It will expand the capabilities to medium or medium-high levels along the three dimensions as described in Figure 4 (around 3-4). Clearly there are even more complex tasks that the agent will not be able to learn. For example, a task such as cooking a full meal or driving a taxi would require very high level capabilities along these different dimensions. Human task learning and performance abilities go even further than those listed in the table.

8. Evaluation

A major claim of this dissertation is that this work expands the space of tasks that an ITL agent can learn along these three dimensions of task complexity beyond both previous work and other related work. Characterizing this space of learnable tasks is challenging in and of itself, and a goal of this dissertation is to provide a framework for better defining and comparing what tasks different approaches can learn through Interactive Task Learning. The evaluation of this claim will consist of two major parts: demonstrating the agent's task learning abilities through experiments and comparing its capabilities to other related work. The following sections describe the different methods and experiments that will be used when evaluating this dissertation.

8.1 Evaluation Domain

Fully exploring the agent's task learning capabilities requires a domain that can afford a large range of different tasks. Previous work with Rosie has demonstrated that it is able to learn tasks on real-

world robots and deal with lower-level issues such as object tracking errors or the arm dropping a block. However, the capabilities of the robot platforms available to our lab are still limited, and would restrict the tasks that could be learned. The focus of this work is to learn high-level task representations and relies on a known set of primitive actions and stable perceptual capabilities. Therefore the experiments listed here will be performed in a simulated environment, which will afford a broader set of actions and more complex behavior. For example, the simulator supports actions that involve interactions between objects, such as heating up a cup of water in the microwave. Using a simulator will ignore problems and errors that can arise in real-world robotic perception and manipulation. However, it will enable the agent to learn higher level tasks and will better explore the task learning capabilities that this dissertation focuses on.

Most of this work will be tested and evaluated using a simulated mobile robot environment based on the Ai2Thor agent simulator (Kolve et al., 2017). This simulator supports realistic looking 3D environments with multiple rooms, different tools and appliances, and containers. For example, it has a microwave with a working door, which can hold objects and heat them up when turned on. However, the agent controller is fairly simple, with discrete actions that do not involve motor planning, such as `pick-up(apple)`, `turn-right`, `open(fridge)`, and `use(knife, apple)`. The agent does have a realistic sense of perspective, where it only sees items in its view region that are not in a closed container, but object detection and recognition is perfect. We plan to do experiments in both an office environment and a household environment.

8.2 Case Studies

Case studies are a valuable tool to show how the agent learns a specific task and describe the internal processing and learned knowledge. These can demonstrate that the agent does successfully learn a task, although it is limited a single instance. They are more useful to bring clarity to how the approach works and what is learned. They can also highlight cases where our integration of these new task learning capabilities in a unified manner leads to interesting interactions between them. For example, in Mininger and Laird (2018), we described a case where we removed the model for one of the actions, and the agent still learned a task by blending both procedural and goal-based formulations.

8.3 Qualitative Analysis

As described earlier, a focus of this work will be developing a framework to characterize the space of tasks that an ITL system can learn. This will include a number of dimensions of task complexity, some of which will be related to the three main areas of diversity in this dissertation. We will perform an analysis of our approach according to this framework, making sure to identify how it improves beyond the previous version. For these different dimensions, we will construct a set of tasks of varying complexity and show which our agent learns. It will be a manually guided exploration of where the task-learning boundaries to our approach lie.

Deliver the package to the main office.
Deliver the apple to the living room.
Deliver three sodas to the kitchen.
Deliver books to the conference room until there are four books in the conference room.
After the microwave is off, deliver the hot water to Alex's office.
Deliver a paper ream to Bob's office (ream not in starting room).

Figure 5: Example of teaching one task and then possible test variations

8.4 Task Transfer Experiments

One of the most significant strengths of our approach is its ability to generalize from one task instance and transfer that knowledge to many other variations on that task. When we claim that the agent successfully learned a task, we do not only mean the agent can replicate that same task again in the future, but that it can perform it with different objects, modifiers, and starting conditions without needing further instruction. The agent may need to initiate further instruction if a task variation involves additional knowledge not taught in the first instance, though it is valuable to see when the agent is able to acquire that knowledge on its own. For example, in one variation it may need to do a `find` action that it did not do in the training task, and it may be able to identify that through internal reasoning. However, if the task involved a new location, the person would likely have to show the agent where it was. Since the ability to generalize is a crucial claim of this work, it is important to evaluate how well it transfers to other variations.

To this end, we will perform transfer experiments where we will give one training instance of a task, then ask it to perform a set of task variations. This set will include randomly generated starting conditions, task arguments, and modifiers. An example of possible variations can be seen in Figure 5. We will measure both whether it was ultimately successful at performing the task and how many additional instructions it needed.

8.5 Replication Studies

A significant challenge in evaluating work in ITL is that there are no common data sets, task lists, benchmarks, or environments that can be used to compare different approaches. One contribution of this dissertation will be to develop a framework to describe the space of task complexity which can be used to compare different approaches and the kinds of tasks they can learn. In order to demonstrate that our approach is a significant improvement to the field of ITL in terms of increased task complexity, we will perform a detailed analysis of related work, quantify their task learning with respect to our framework, and where possible, replicate their task learning performance. The goal is to identify to what extent our approach can learn the tasks that they describe, and which of our tasks they cannot learn. When comparing approaches, the main evaluation criteria will be what percentage of their described tasks can our agent learn, which tasks of ours could their approach not learn, and comparing the number of instructions it takes to teach each task. Below are some

examples of possible candidates to use for this replication study, and some predictions of to what extent this agent could learn the tasks that they describe.

8.5.1 She et al. 2014

In this work, after the instructor leads the agent through the task, the agent compares the initial and final states to extract a goal representation. It can then use planning to achieve the goal in future executions. The learned tasks include block manipulation ones such as *Pick up*, *Move*, and *Stack*. We should be able to fully replicate these tasks.

8.5.2 Mericli et al. 2014

This approach learns a task as an Instruction Graph, where the task procedure is explicitly represented as a graph with edges representing control flow. It can include constructs like loops and conditionals. There is minimal ability to generalize or to plan, as the procedure is fixed. In the paper there is an example task of getting coffee in an office environment, which we should be able to fully replicate. However, there might be some more complex tasks involving loops that our approach couldn't learn.

8.5.3 Mohseni-Kabir et al. 2015

In this work, the agent learns an HTN in a bottom up manner, where the instructor gives low level commands and can then combine these to form higher level tasks. The agent is taught the task of rotating tires in a car mechanic domain, where it must unscrew and remove all the tires, rearrange them in a particular way, and screw them back on. This domain might be somewhat harder to replicate, as it involves a subpart hierarchy, but the task structure itself should be possible.

8.5.4 Suddrey et al. 2017

This approach learns an HTN, and for each task learns its preconditions, postconditions, task decomposition, and argument mappings between the task and its subtasks. During execution, it can interleave a forward planner so it can adapt to different starting conditions and have some plan recovery. They also can do tasks that involve sets of objects, such as clearing all the items from a table. In the paper, it describes experiments with a Baxter robot and a simulated domain, with tasks such as *Pick Up*, *Move*, *Put Away*, and *Clean*. We should be able to fully replicate these tasks.

9. Future Plans

Significant progress has already been made for the first two types of diversity: action types and task formulations. Many of the actions described in section 4 (e.g. find, say, ask, remember, recall), have been implemented and were included in a 2016 ACS publication Mininger and Laird (2016). This involved significant work in extended the agent to learn tasks in a partially observable, multi-room environment. For diverse task formulations, the agent already learned goal-based formulations, and I have since added the ability to learn procedural formulations Mininger and Laird (2018). There is still work to be done with adding some other minor variations of these (such as composite or

negative goal tasks), and adding in a simple reinforcement learning formulation, but this should be relatively simple given what has already been done. The most significant piece of work left is in the third type of diversity: task modifiers.

My plan is to start working on implementing additional task modifiers directly after my thesis proposal, and to target a 2019 IJCAI paper with a February deadline. Part of this work will also include further developing the simulation environment and setting up an evaluation domain, which will be directly valuable to this dissertation. After submitting the IJCAI paper, I will start on writing my thesis. I am hopeful much of the groundwork for the thesis will be laid during my work on the IJCAI paper, especially the simulation environment. I expect the rest of the winter and spring to be spent on finalizing the work on the three major sections and writing them, which I expect to take four to six weeks each. Thus the bulk of the thesis work will be finished by early summer. From there, final writing should take another few months, with the goal of defending by the end of summer 2019.

References

Akgun, B.; Cakmak, M.; Jiang, K.; and Thomaz, A. L. 2012. Keyframe-based learning from demonstration. *International Journal of Social Robotics* 4(4):343–355.

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Plow: A collaborative task learning agent. In *AAAI*, volume 7, 1514–1519.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Arumugam, D.; Karamcheti, S.; Gopalan, N.; Wong, L. L.; and Tellex, S. 2017. Accurately and efficiently interpreting human-robot instructions of varying granularities. *arXiv preprint arXiv:1704.06616*.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI*, volume 2, 1–2.

DeJong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine learning* 1(2):145–176.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

Frasca, T.; EDU, T.; Oosterveld, B.; Krause, E.; and Scheutz, M. 2018. One-shot interaction learning from natural language instruction and demonstration. *Adv. Cogn. Syst* 6:1–18.

Gemignani, G.; Bastianelli, E.; and Nardi, D. 2015. Teaching robots parametrized executable plans through spoken interaction. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 851–859. International Foundation for Autonomous Agents and Multiagent Systems.

Gluck, K. A.; Laird, J.; and Lupp, J. 2018. *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks Through Natural Interactions*. MIT Press.

Howard, T. M.; Tellex, S.; and Roy, N. 2014. A natural language planner interface for mobile manipulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 6652–6659. IEEE.

Huffman, S. B., and Laird, J. E. 1995. Flexibly instructable agents. *Journal of Artificial Intelligence Research* 3:271–324.

Kolve, E.; Mottaghi, R.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Laird, J. E. 2012. *The Soar cognitive architecture*. MIT press.

Lallée, S.; Yoshida, E.; Mallet, A.; Nori, F.; Natale, L.; Metta, G.; Warneken, F.; and Dominey, P. F. 2010. Human-robot cooperation based on interaction learning. In *From motor learning to interaction learning in robots*. Springer. 491–536.

Lignos, C.; Raman, V.; Finucane, C.; Marcus, M.; and Kress-Gazit, H. 2015. Provably correct reactive control from natural language. *Autonomous Robots* 38(1):89–105.

Lindes, P., and Laird, J. E. 2017. Ambiguity resolution in a cognitive model of language comprehension.

Matuszek, C.; Herbst, E.; Zettlemoyer, L.; and Fox, D. 2013. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, 403–415. Springer.

Meriçli, C.; Klee, S. D.; Paparian, J.; and Veloso, M. 2013. An interactive approach for situated task teaching through verbal instructions. In *Intelligent Robotic Systems*, 47–52.

Mininger, A., and Laird, J. E. 2016. Interactively learning strategies for handling references to unseen or unknown objects. *Adv. Cogn. Syst* 4:1–16.

Mininger, A., and Laird, J. E. 2018. Interactively learning a blend of goal-based and procedural tasks. *Ann Arbor* 1001:48109–2121.

Misra, D. K.; Tao, K.; Liang, P.; and Saxena, A. 2015. Environment-driven lexicon induction for high-level instructions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, 992–1002.

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

Mohan, S., and Laird, J. E. 2014. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *AAAI*, 387–394.

Mohseni-Kabir, A.; Li, C.; Wu, V.; Miller, D.; Hylak, B.; Chernova, S.; Berenson, D.; Sidner, C.; and Rich, C. 2018. Simultaneous learning of hierarchy and primitives for complex robot tasks. *Autonomous Robots* 1–16.

Mollard, Y.; Munzer, T.; Baisero, A.; Toussaint, M.; and Lopes, M. 2015. Robot programming from demonstration, feedback and transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Petit, M., and Demiris, Y. 2016. Hierarchical action learning by instruction through interactive grounding of body parts and proto-actions. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 3375–3382. IEEE.

Phillips, M.; Hwang, V.; Chitta, S.; and Likhachev, M. 2016. Learning to plan for constrained manipulation from demonstrations. *Autonomous Robots* 40(1):109–124.

Rybski, P. E.; Stolarz, J.; Yoon, K.; and Veloso, M. 2008. Using dialog and human observations to dictate tasks to a learning robot assistant. *Intelligent Service Robotics* 1(2):159–167.

Rychener, M. D. 1983. The instructible production system: A retrospective analysis. In *Machine Learning, Volume I*. Elsevier. 429–459.

She, L., and Chai, J. 2017. Interactive learning of grounded verb semantics towards human-robot communication. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 1634–1644.

She, L.; Yang, S.; Cheng, Y.; Jia, Y.; Chai, J.; and Xi, N. 2014. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, 89–97.

Simon, H. A., and Hayes, J. R. 1976. The understanding process: Problem isomorphs. *Cognitive psychology* 8(2):165–190.

Suddrey, G.; Lehnert, C.; Eich, M.; Maire, F.; and Roberts, J. 2017. Teaching robots generalizable hierarchical tasks through natural language instruction. *IEEE Robotics and Automation Letters* 2(1):201–208.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M. R.; Banerjee, A. G.; Teller, S. J.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, volume 1, 2.

Winograd, T. 1972. Understanding natural language. *Cognitive psychology* 3(1):1–191.