SAL: a Self-Aware Learning system

by

Tristan Andrew Fraser Thrush

B.S., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Author .....
Department of Electrical Engineering and Computer Science
August 23, 2019

Certified by .....
Patrick Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Certified by .....
Randall Davis
Professor of Computer Science and Electrical Engineering
Summer Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

SAL: a Self-Aware Learning system

by

Tristan Andrew Fraser Thrush

Submitted to the Department of Electrical Engineering and Computer Science
on August 23, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I take a step towards understanding how and why humans learn to solve problems about their solving of problems. I present a general-purpose neural reinforcement learning system called SAL, which can learn to think about its own problem solving, and use this capability to learn how to solve problems at another level. I show that SAL can use self-reference to articulate, and learn to articulate, its thoughts to a human, and internalize and apply a human's help, in natural language. I also demonstrate that SAL's abilities are enabled by an internal representation that shares important properties with, and is easily converted between, natural language.

On the practical side, I argue that SAL can inform production question answering systems research. SAL can answer multi-step questions that are grounded in the world by extracting operational knowledge from pre-trained word embeddings. As an example, SAL knows how to use the action associated with "grab [the] diesel jug" to get closer to a solution, given the state of a physical world and a goal. And SAL can do this without any actual experience using (and without ever being told by a human about) any action associated with "grab" or the argument "diesel jug." SAL can do so with both very little training reward data and without assuming anything about the operational meaning of a particular lexical item, or composition of them, at first. Alternatively, typical neural reinforcement learning systems can not learn like SAL; they only work with a level of data that would be difficult to achieve in the real world.

SAL's implementation, trained models, analysis code, and instructions, are at `https://github.com/TristanThrush/sal`. It is easy to add new problems (even in new domains) that you want SAL to learn.

Thesis Supervisor: Patrick Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Summer Thesis Supervisor: Randall Davis
Title: Professor of Computer Science and Electrical Engineering

Acknowledgments

I am incredibly fortunate to have met Professor Winston. He introduced me to artificial intelligence as a freshman, and mentored me through the years from that point onward. His perspective has had a profound impact on the way I think about artificial intelligence and conduct research.

I am grateful to Professor Davis, who offered countless hours of fantastic support and expertise as another thesis supervisor when Professor Winston became ill.

Finally, I am thankful for my father, mother and step-father, Jerry Thrush, Arlene Fraser-Fuss, and Eric Fuss, who have continued to support me, along with the rest of my friends and family. Jerry Thrush also provided editing assistance. It takes a village.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order to understand human intelligence, we need to understand the capacity of humans to recursively solve problems about their own problem solving. Particularly, it is important to understand how and why we achieve part of this feat with thoughts that can be translated to and from natural language. For example, a person can answer a question about how they answered, or are currently answering, another question. They can also answer a question about how they answered the question about the original question and so on. Among other examples, someone can remember and provide steps that they tried while solving a problem, and then articulate the problem and their solution attempt to another person so that they can later internalize that person's help, or simply keep the information as interest to themselves. Someone can also explain that they already explained their solution and so on. While this introspective capacity can be used to simply explain oneself when provoked by other people, I postulate that the use of such self-referential operators greatly increases someone's ability to learn how to solve a problem, particularly (but not only) because it enables them to receive help from someone (verified in Chapter 3). A learning apparatus can also enable the learning of how to use these operators in the first place. I present some solutions that are unattainable without certian self-referential operators (also in Chapter 3).

In this thesis, I explore the human capacity to pose self-referential operators and solve self-referential problems that they induce, how learning both benefits from and

helps guide this capacity, and the humanlike behaviors that emerge from this framework. I find this topic particularly exciting because it deals with the interface between introspection (which I consider a component of self-awareness) and learning (of which, some types are associated with perception, pattern-matching, and subconscious thought). In order to understand these aspects of human intelligence, I have constructed a general-purpose reinforcement learning framework, called SAL (a **S**elf **A**ware **L**earning system), that implements them. I am interested in both the engineering benefits to the field of automated question answering and the scientific prospect of developing a system that shares (what I believe to be) a key component of what makes us human; I argue that SAL is of both theoretical and practical interest. In the following sections, I explain my research vision, provide the necessary background to understand the body of this thesis, and present an example of SAL's capabilities.

Before reading further, note that I do not provide rigorous definitions for loaded words such as "introspection" and "self-awareness," because I am not sure what the definitions should be, and they are not of critical importance to my thesis. Minsky (1986) referred to such terms as "suitcase words:" our society has packed so many meanings inside each word that it is hard to construct a definition that everyone agrees with. Instead, I provide examples of processes that I hope to model, such as that in Section 1.4.

1.1 Inner Language as an Enabler of Self-Aware Cognition

Because I purport that SAL can think about its own thoughts, it is important to discuss different types of thoughts, how they might be represented in the human mind, and how my system models them. Note that a person's thoughts can trivially be divided into two types: where introspection (at least the kind that can be externalized via natural language) is possible versus impossible.

A person cannot always articulate their thought processes. Someone may be able to recite some steps that they took to pick up an object (such as walking, reaching, and grasping), but they "ground out" in operators that are beyond the explanation barrier. For example, one cannot reflect on how they controlled their arm to follow a motion plan or the specific joint angles in the plan: their arm just "did it" at a certain point. Someone might be able to conduct extensive investigations on their brain to determine exactly how mental modules generate specific joint angles, but that is different than the scenario where the modules tell such introspective parts of the mind the angles and derivations directly.

How are a person's thoughts represented before they "ground out" in the sub-introspective portions of the mind? One hypothesis is that such thoughts are represented as a general-purpose inner language that is usually translatable between, and shares important properties, with natural language (Winston, 2012). Natural language presumably evolved so that we can use it to share thoughts with each other, so, under this scenario, there is a strong relationship between the two.

I do not suggest that language and thought are the same thing. Fedorenko and Varley (2016) found that there was insignificant activity across known language regions of the brain when humans solved a variety of problems involving logic, addition and subtraction, thinking about others' thoughts, music, and navigation. It is also evident that global aphasics who cannot internalize or produce language can still solve such problems. A theory that is consistent with these findings, however, is that there is an internal language that is used in general-purpose problem solving, and there are internalization and externalization engines that are usually capable, in healthy individuals, of translating the two representations (natural and inner language) between each other.

1.1.1 An Implementation

In my experiments, SAL uses a symbolic language which I call Innerese, because it is similar enough to the language that the Genesis Group (Winston & Holmes, 2018) calls Innerese. Innerese resembles symbolic logic from linguistic theories of semantics

(Altshuler, Parsons, & Schwarzschild, 2019). Fodor's Language of Thought Hypothesis (1975) suggests that humans parse natural language into logical expressions, which form a model for the inner language of thought that I have discussed.

The Genesis Group's Innerese is implemented in the Genesis system, which uses the START natural language system (Katz, 1997) as a step in parsing English into this representation (which Genesis can then convert back to natural language). START analyzes natural language by breaking it into sequences of ternary expressions that have the form <relation subject object>. Although other types of expressions are possible in Genesis's model of inner language, ternary expressions are the fundamental building blocks and can be hierarchically nested in other ternary expressions. As an example, Genesis's Innerese parse of "I want to have pie" could be "<want I <have I pie>>." It is difficult to define Genesis's Innerese precisely, because its workings are obscured by the large body of code in Genesis, and Genesis has a variety of bugs.

The code for SAL is completely detached from Genesis and START; it takes as input an approximation to Genesis's Innerese and uses a re-implemented system that translates it to and from natural language. I provide several examples of the Innerese representations that SAL uses throughout the body of this thesis, along with definitions of how specific primitives are composed with constituents (see Figure 1-1 and Figure 3-1).

Note that, while I commit to a specific grammar for my experiments, the general SAL framework does not rely on anything other than consistent hierarchical parses, so the Innerese implementation that I use in this thesis could be replaced with a variety of grammars. I believe that it is useful for the reader to keep this statement in mind so that they know that there is nothing hidden in something very specific about the symbolic language that I use.

SAL does not only have access to scope knowledge given by the hierarchical structure from an Innerese input. SAL also has some amount of lexical knowledge given by word embeddings for each word (Pennington, Socher, & Manning, 2014), as Innerese words happen to be represented by their English counterparts. Word embeddings are high-dimensional vectors for each word in a dictionary; they are typically learned

without supervision from a large amount of co-occurrence data in a natural language corpus. The result is that the vectors for words that humans consider semantically related are typically close to each other (using a variety of distance metrics) in the vector space.

1.2 Solving Problems in an Inner Language

Winston's Self-Aware Problem Solver (Winston, 2018) inspires a component of SAL; it is an infrastructure where operators can be programmed to manipulate the state of a problem and check goal conditions, represented in Innerese. The main idea of the Self-Aware Problem Solver framework is that operators can update the Innerese state of a problem, and add Innerese expressions that explain the problem solver's thought process, even though they do not give a comprehensive account of what they did. Consider the example from the previous section where a person can articulate that they have grasped an object but does not know exactly what joint angles were formed in their motion plan. If operators are provided that can update the Innerese problem state representation by adding explanations, and possibly changing goal conditions, indefinitely, then the problem solver would be enabled to solve problems containing its explanations (e.g., self-referential problems) over and over again, as much as memory limitations allow for. And due to the Innerese representation, it can easily be converted to natural language and articulated. These are reasons why I consider the problem solver worthy of the title "Self-Aware," in the sense that the behavior it can achieve is an important component of self-awareness.

To the best of my knowledge, this thesis is the first to provide operators that enable the Self-Aware Problem Solver framework to recursively reason about its own reasoning indefinitely, to enable the learning of how to use these self-referential operators, and to enable the learning of how to use these operators to learn. The operators are provided in Section 2.2.

1.3 Learning to Solve Problems in an Inner Language

The Partial Mental State Inducer (Thrush & Winston, 2018) extends the capabilities of Winston's Self-Aware Problem Solver by enabling it to learn how to best apply its operators depending on the type of problem that it faces. It was the first system that took an Innerese representation of a problem scenario as input and used learned associations from this representation to advise on which operators to use.

The Partial Mental State Inducer has limitations, which are resolved in this thesis. First, it does not use the kind of self-referential operators that I have discussed to recursively create self-referential problems about current problems and it does not learn to use them or use them to help learning. Also, PMSI uses non-compositional operators. For example, PMSI used an operator that found and combined all like terms in an equation. Contrast this with a hypothetical compositional operator such as "add(1, subtract(3, 2))." I address the issue of learning to choose compositional operators in this thesis, as I believe that this has been a major qualitative difference between the capabilities of PMSI and humans. On the practical side, my hope is that one day a system such as SAL will scale easily to unseen domains without any expert-coded operator capabilities and just with learning. Ideally, SAL would use very primitive hard-coded capabilities (to handle e.g., perception and logic); other operator capabilities need not be programmed but can instead be learned compositions of these symbol manipulation and perception primitives.

1.4 Example

In this section, I provide an example of the behavior that I envisioned, within a spatial manipulation domain. The example is a real problem that SAL learns to solve. The rest of the thesis is dedicated to understanding how SAL accomplishes such feats and examining them in more detail.

SAL is given a virtual body with two grippers, which can hold one item each,

and learns by experimenting with its operators in a simulated world. The state of the simulated world is communicated to SAL as Innerese,[1] and a goal is also given to SAL in Innerese. I explain in Chapter 2 how SAL also considers the Innerese representation for large numbers of candidate compositional operators that it could take, and learns an evaluation function to judge which composition is best, given the Innerese problem's state and goal. Throughout this thesis, I use the term **operator predicate** to refer to an operator template that specifies arguments, and can be instantiated to form an operator (I do not use this term to mean a function which returns a Boolean). In this domain, SAL has two operator predicates that it can use to move items: **Pick(x)** and **Place(x,y)**, which are rigorously defined in Chapter 3, but can be understood for now by their names. Also, SAL has a collection of a few self-referential operator predicates defined in Section 2.2; it can instantiate, and learn to use them, just like any other operator predicates. SAL initially only knows that **Pick(x)** takes one argument and **Place(x,y)** takes two arguments. SAL can try to use **Pick(x)** and **Place(x,y)** with any constituent in the Innerese representation of the problem's state and goal, but the simulator is programmed such that **Pick(x)** will only change the state of the simulated world if its argument is an Innerese constituent representing a movable object, and **Place(x,y)** will only do so with that of a movable object as its first argument and that of any object in the world as its second argument. Of course, a well-formed **Pick(x)** or **Place(x,y)** still may not be useful to solve a particular problem. To enable learning, **Pick(x)** and **Place(x,y)** also check the goal conditions and return a reward signal that depends on whether the goal is met. To clarify, the simulator is programmed to know which objects are movable, but SAL must learn this information, and SAL must also learn an operator, or sequence of them (along with how each updates the state and possibly goal of the problem), to solve for a given goal.

In the specific problem that SAL learns to solve, the Innerese initially expresses that there is an oil jug on a workbench and a tire on a stool and that the goal is for

[1] More generally, inner language need not come from a natural language parse, but could come from perception.

the oil jug to be on the stool. A sequence of the real Innerese states and goals are given for this problem in Figure 1-1, paired with the real Innerese representation of SAL's learned solution steps (the roots of Innerese parses for the actions that SAL chooses are circled in red).

SAL learned by solving the problem just 40 times. This means that every time a goal state was reached, the problem started over to be solved again, 40 times. So, SAL only received 40 positive reward signals. Again, SAL is a reinforcement learning system and so it trains by experimenting with operators (in a somewhat random fashion at first) to solve a problem several times, observing actions, state transitions, and rewards. For example, on the first problem-solving episode, SAL chose and applied 14 operators which all returned negative rewards, only one of which (given by "(pick (oil jug)") was useful at actually getting SAL closer to a solution. SAL then applied another useful operator that returned a positive reward, indicating that the problem was solved (given by "(place (on (oil jug) stool))"). Then the problem was reset and the process continued. SAL improved through the training epochs by getting a progressively better idea about which actions were actually useful. At the end of the 40 epochs, SAL's network update procedure had sifted through all the operators that it tried at first, to isolate the sequence of two that actually solved the problem. I could then evaluate SAL's performance by having it solve the problem in essentially the same way that it did during training.

In every step during the learning process, the branching factor had about 200 possible individual **Pick(x)** and **Place(x,y)** argument instantiations that SAL could choose from, only one of which advanced the solution. There were also an infinite number of self-referential operators that SAL could take (these operators have yet to be defined, but assuredly do not provide any extra manipulation capabilities). By the end of training, SAL learned a solution that minimizes the number of total movements (again, given in Figure 1-1), which is a difficult task considering how little SAL assumes at first. A version of SAL without self-referential operators could not escape the issues imposed by such a large branching factor to learn a solution, or even a partial solution. This is an important point: SAL learned to use its self-

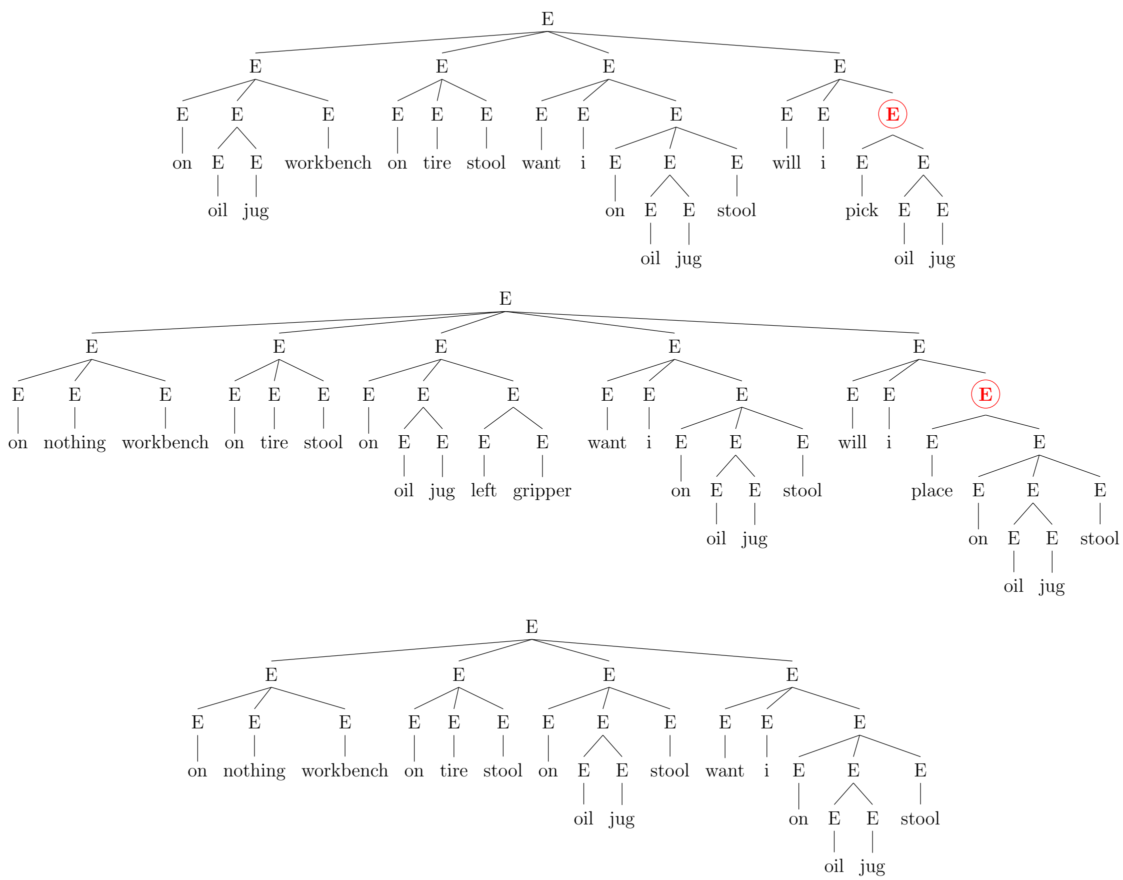

Figure 1-1: Innerese representations that SAL uses in solving a problem. SAL uses the Innerese representation of a problem's state and goal, and learning through experimentation, to select a constructed operator (also represented in Innerese) to apply. An operator can update the state of the problem (and possibly the goal), and SAL repeats the process until the problem is solved. The objects represented by E are embeddings. Those directly above the words are pre-trained word embeddings (e.g., E - tire). The others embed the corresponding constituents below. The constituent embeddings contain information that allows SAL to assess how good an operator is, when paired with the problem's state and goal. I explain later in the thesis how the constituent embeddings are learned. Note that SAL could have also used an operator that inserts self-referential information into the Innerese for the state of the problem, such as "(remember i externalize)" (to express a memory of a previous action). SAL indeed learns to insert this self-referential information to help it learn in the first place (as will be discussed in Chapter 3), but does not insert this information in its final learned solution.

referential operators to internalize natural language help from a human as Innerese and carry it out. SAL additionally learned to externalize its Innerese problem's state and goal in natural language so that someone can help in the first place, including mentioning important actions that it took. It is also important to note that, on many occasions the human input to SAL was of no use in getting closer to a solution, and on no occasion was the full solution given to SAL. And no help was given when SAL finished training and was evaluated; help was not necessary by the end because a natural tendency emerged for SAL to not ask for help as it became more sure of the correct solution. A real natural language externalization that SAL gave during the learning process was "nothing on workbench. tire on stool. oil jug on left gripper. i want oil jug on stool." And a real instance of help given to SAL as a response to this externalization was "place oil jug on stool," which SAL then parsed into Innerese to know the explicit argument hierarchy (e.g., is it "(place (on (oil jug) stool))" or "(place oil (on jug stool))" or an alternative?). The operator corresponding to the Innerese representation was then realized and carried out to solve the problem.

SAL also learned at least part of the compositional semantics of Innerese and useful generalizations that can be extracted from the lexicon. It learned that **Pick(x)** is useful when its argument is neither "oil" nor "jug" but the compositional "oil jug." It learned this by experimenting with the simulator, which knows that "(oil jug)" is a movable object, but the component words are not. It also learned how to apply its operators with the appropriate arguments on somewhat new problems containing objects and goals that it had no experience with; it knew that it should attempt to move objects such as "petroleum jug," "diesel jug," and that it should not attempt such actions with the component words. It also learned how to apply **Pick(x)** and **Place(x,y)** when they were represented by words in Innerese that are semantically related to "pick" and "place," such as "grab" and "set." To clarify, I mean that the words in Innerese representations in Figure 3-1 were replaced but the structure was kept the same. SAL performed worse, but well above chance, with replacement words that have no obvious semantic relationship to "pick" and "place," such as "guiltiest" and "phishers" (a seemingly strange generalization, but reasonable if SAL really has

no other information but the syntax to go by).

Attention should be drawn to an important aspect of Innerese: it is so close to natural language that it enables expression and internalization to and from natural language. Among other abilities that Innerese provides, SAL can receive natural language help, and SAL can use some amount of self-reference to explain its problem and memories in the first place. A system could have tried to learn an inexplicable policy directly from image inputs, instead of Innerese inputs. But it is far less obvious how this representation would provide the abilities that Innerese enables.

It is possible that my research vision and example evoke memories of STRIPS (Fikes & Nilsson, 1971). But my system actually has different goals, and has far fewer requirements. STRIPS assumes that there is a world model to conduct search in, and SAL does not conduct any search in a world model to suggest operators. SAL learns to suggest operators via cues from the Innerese representation of the problem, and this learning need not be from a model of the world (it could, in principle, be from the real world). Alternatively, a search-based system such as STRIPS must always use some model of the world to take actions within. SAL also does not assume initial knowledge of what operators do or even full knowledge of the problem; it can learn what the operator use conditions are, handle only partial observation of the problem state, and handle operators with a probabilistic behavior. Importantly, STRIPS cannot think about its own thinking like SAL. As an example, STRIPS cannot learn from natural language help and cannot explain its own problem solving by converting its internal representations to natural language.

Chapter 2

Architecture

SAL's architecture can be used to explore the humanlike abilities that are enabled by Innerese: one of which is a presumed aspect of self-aware thought that can be translated to and from natural language, and one of which is a general-purpose representation for learning with. It can also be used to explore how these two capabilities connect with each other to transcend their parts: the learner can help the system to apply its self-referential operators to advance solutions, and the self-referential operators can aid learning in the first place.

In the following section, I define the learning problem specifically. I then describe the modules that enable learning, the architecture of the learner, and how the learner updates the network while training. In Section 2.2, I describe the self-referential operators and how they take the system to another level.

2.1 Learner

The task of my learning apparatus is to model the human capacity to select compositional operators based on learned cues given by the problem at hand. It is able to learn how to solve general problems expressed in Innerese through experimentation. It generates and chooses from a possibly infinite set of compositional operators, predicting what effect such an operator will have on subsequent states, and chooses another,

until a solution is reached.[1] Some of these operators might change the problem's state or goal, which might be reflected in an update to the Innerese that represents the problem. And, to enable learning, all operators return a reward signal.

I developed a novel neural reinforcement learning system where the problem's state and goal, and a proposed operator, are all considered together, as Innerese. Throughout this thesis, I refer to a combined problem state, problem goal, and proposed operator, all expressed in Innerese, as a **mental condition**. The system takes this Innerese as input, and outputs a single value that is the learned anticipated long-term reward of the mental condition; I define this value precisely with a novel loss function in Section 2.1.3. The important idea to know at this point is that a higher value estimate for a mental condition means that its operator is expected to be better at advancing a solution, given the problem's state and goal. SAL chooses from a set of mental conditions produced by a generator; these mental conditions are each composed of the existing problem's state and goal and a different constructed operator. SAL chooses the mental condition that has the highest value estimate, and carries out its operator. In essence, choosing a mental condition equates to choosing an operator, but SAL needs the problem information from rest of the mental condition to estimate a value; this is why I sometimes say that SAL chooses a mental condition. I also say this to distinguish SAL from typical neural reinforcement learning systems, such as Q-value learners, that take in a state and goal, produce value estimates for a list of actions, and choose the one with highest estimate (Mnih et al., 2015). As I explain in Section 2.1.3, it is strange to formalize SAL's task as typical Q-value learning, because representations of SAL's proposed actions are part of its input, and are indistinguishable at first from problem state and goal information; operators are also not chosen from a finite or necessarily fixed set.

2.1.1 Operator Generation

How exactly are the proposed mental conditions generated?

[1] Choosing from a large set of operators via learned cues is not the same as search by applying operators within a world model.

To generate a compositional operator, the generator takes a set of given operator predicates and also the set of all Innerese constituents in the Innerese for the problem's current state and goal. As an example, for the Innerese "((on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)))," the set of all constituents is "{((on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool))), (on tire stool), (on (oil jug) (left gripper)), (want i (on (oil jug) stool)), (on (oil jug) stool), (oil jug), (left gripper), on, tire, stool, oil, jug, left, gripper, want, i}." Each operator predicate lists what arguments it requires, which are either Innerese constituents or operators. A predicate also specifies an Innerese template for corresponding instantiations of the operator to follow. For example, the Innerese template for **Place(x,y)** (from Chapter 1) is "(place (on x y))" where x and y must be some Innerese constituent. Operator predicates can be thought of as roughly analogous to predicates in linguistic theories of semantics (Altshuler et al., 2019). A predicate is also a name for a function that returns a Boolean value; I do not refer to this type of predicate.

The generator starts with an operator predicate selected uniformly at random and instantiates each of its arguments with an operator or an Innerese constituent, depending on what the arguments require. If an argument calls for an operator, then the generator recurses and instantiates this operator with the same procedure. If an argument calls for an Innerese constituent, then one is selected uniformly at random from the set of all Innerese constituents in the problem and goal. The generator has finished generating an operator when there are no more arguments to fill. For example, the Innerese for a generated compositional operator (using the constituents given in the example in the above paragraph) could be "(then (pick tire) (place (on (oil jug) stool)))." In that example, the given operator predicates have the following Innerese templates: "(then a b)," "(pick c)," and "(place (on d e))" where a and b are Innerese for other operators and c, d, and e are Innerese constituents from the problem and goal. The generator repeats this process until a specific number (which is a coded hyperparameter - about 500, in many of my experiments) of compositional operators have been generated, so that SAL can then

pick the best one from this set.

Even though many generated operators will fail outright, or prove to be useless, as long as enough productions are generated, there should be a compositional operator that turns out to be very useful; I discuss this idea in more detail in Section 2.1.3. It is the task of the learning system to select that very useful operator.

See Figure 2-1 for an example of a problem, and possible generated operators to tackle the problem. The parse trees represent possible inner language structures, and they are not actual Innerese parses. The reason for this is to show that the applicability of the general ideas behind SAL, and the implementation of the current generator, go beyond the specific Innerese grammar that I use to any hierarchical grammar that models the presumed inner language. The structures for the generated operators have a root that is circled and red. Some of these generated operators might solve the whole problem in one step (e.g., the one that instructs walking to the table and moving the arm to the can). Some of the operators might get SAL closer to a solution (e.g., the one that instructs walking to the table). Some of the operators might fail (e.g., the one that instructs moving the can to the table when a movement operator predicate can only move limbs). And some of these operators might not even be convertible to good natural language (e.g., the one that tries to instruct moving "grasp the can" to a location).

To summarize, it is important to note that the actual output of the generator is a set of mental conditions, not a set of operators. The set of mental conditions all have the same problem state and goal, each combined with a different generated operator as in 2-1. The functional definition of the generator is $\mathcal{G}(p, o_p)$ where p is the current problem state and goal Innerese which is combined with each generated operator (and from which the constituents can be found), and o_p is a set of given operator predicates.

2.1.2 Hierarchical Learning

An important feature of human language is that meaning is derived from hierarchical syntactic parses. Hierarchical information is necessary for us to determine the scope

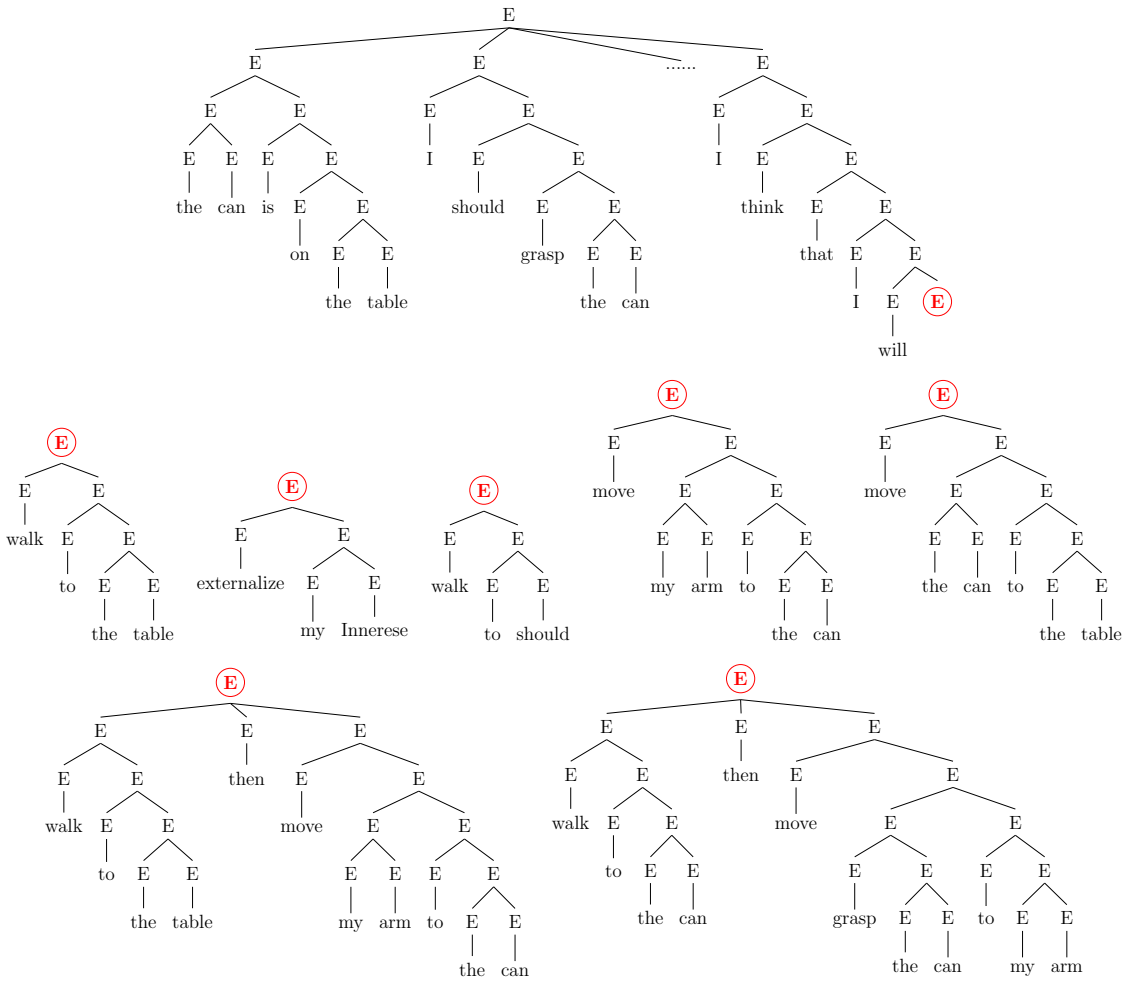

Figure 2-1: Candidate mental conditions, represented as generic parses in this figure to showcase that the architecture of the learner does not depend on any specific choices in the Innerese implementation. The generator generates many possible compositional operators (some of which do not make sense at all). For each of these generated operators, their inner language structure is paired with the structures for the current problem's state and goal (the red roots are each paired with the red leaf in the figure) to form a new mental condition. SAL chooses the mental condition with the highest value estimate, which equates to choosing the best operator, given the state and goal. The value is estimated from the embedding at the root of a mental condition, which is the result of fully merging all of the Innerese constituents (using the merge module in Figure 2-2).

of quantifiers and the nesting of relationships between subjects and objects, just to name a couple of examples. In the Innerese model for inner language, this hierarchical information is represented explicitly. In this section, I detail how I have incorporated this structure into the implementation of the learner that takes in mental conditions and outputs values.

The entire learner within SAL is presented in Figure 2-2. It is a recursive module that merges Innerese constituents into embeddings and then merges those embeddings until an entire mental condition is represented as one embedding. The number at the first index of this embedding is a learned value estimate for the mental condition. This learner is similar to RNNGs and TreeRNNs (Dyer, Kuncoro, Ballesteros, & Smith, 2016), except it assumes that a parse has been provided already (which is Innerese, in SAL's case). RNNGs and TreeRNNs are typically used for parsing or as language models. There are cases where similar networks have been used on existing parses for sentiment analysis (Tai, Socher, & Manning, 2015), but to the best of my knowledge, this is the first use of such a network in an active problem solving scenario where proposed actions represented in a language are also interpreted along with existing language structures that represent the problem to solve.

There are several reasons why the learner is a recursive merge module, as opposed to a standard sequence model that takes in all of the Innerese as one long string. It is conceivable that the merge module can highlight important constituents while suppressing unimportant ones, as in the first example in Figure 2-3. Just as in figure 2-1 (and for the same reasons), this figure also shows a hypothetical model of inner language instead of true Innerese.

In general, symbolic problem solving with compositional statements can involve identifying several constituents whose meanings transcend those of their components. It is easier to do this task when the constituents have already been packed into their own embeddings. An approach where all Innerese word embeddings are taken in as a sequence is less constrained in an undesirable way: it is possible to consider the relationship of individual word embeddings between different sentences and different constituents. Such a scenario is seen in the second example in Figure 2-3.

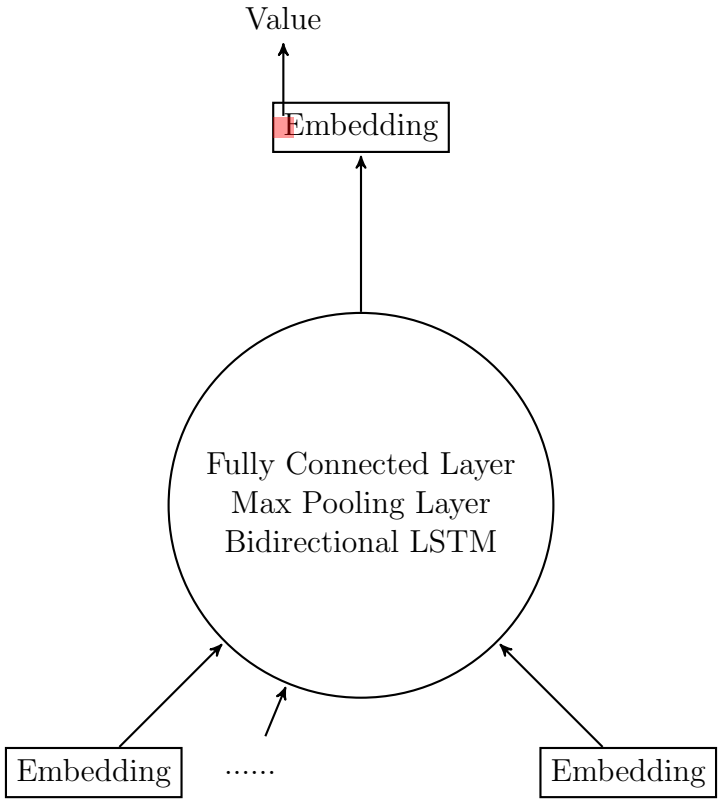

Figure 2-2: The recursive merge module. It traces the hierarchical structure in an Innerese expression. A Bidirectional LSTM (Hochreiter & Schmidhuber, 1997) takes in a sequence of embeddings, which comprise an Innerese constituent, in the forward and backward directions and concatenates the results, and then a Max Pooling (Ranzato et al., 2007) of all of the LSTM hidden state outputs is fed into a fully connected layer. This module merges constituent pre-trained word embeddings to create phrase embeddings. It can then merge these phrase embeddings to create more phrase embeddings or whole sentence embeddings. It can also merge a sequence of several sentence embeddings together to get an embedding for a whole mental condition. Every "E" in Figure 2-1 represents a pre-trained embedding if it is directly above a word, or an output embedding of the merge module otherwise. The bottoms of the parse tree lines indicate inputs to the merge module with the output at their intersection. In the output embedding for the root of a whole mental condition, the number at the first index is the network's value estimate (learned with a loss function defined in Section 2.1.3).

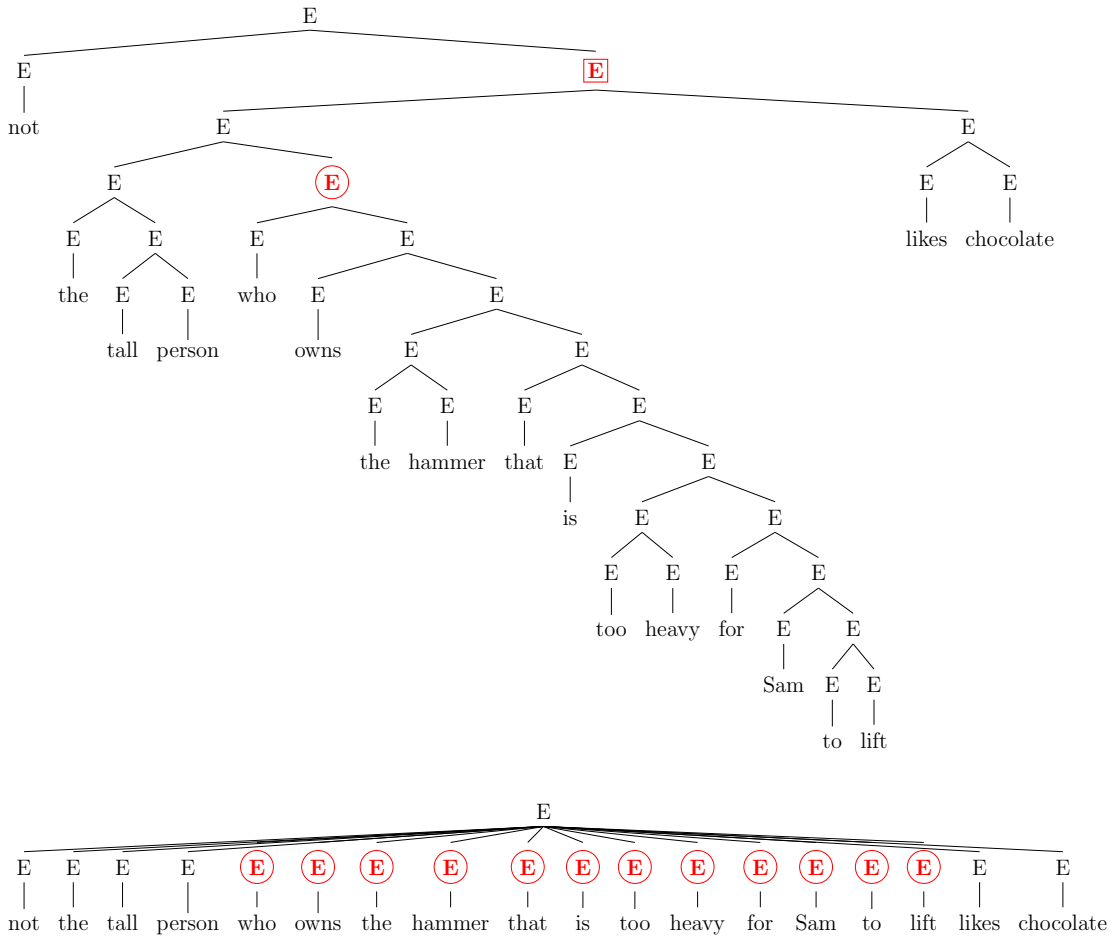

Figure 2-3: As is well known, hierarchical parses are a good way to reduce the complexity of a sentence. First, observe that if one uses the merge module from Figure 2-2, only one embedding rests directly inbetween "the tall person" and "likes chocolate" (circled in red). As a result, it is conceivably easier to associate "likes chocolate" with the correct person. If the information were instead taken in as a sequence (even if tokens to represent the parse structure are provided), there are far more embeddings between the "the tall person" and "likes chocolate," and "Sam" might incorrectly be attached to this trait by virtue of distance. This example shows that the advantages of the merge module go beyond simple sentiment classification; it can help attribute properties to the correct recipient in general cases. As a second example, consider the work that must be done to understand important functions such as "not." To the merge module, the embedding for "not" is combined with an embedding that aggregates everything else in the sentence (rectangled in red). A sequence module has to learn that "not" indeed negates everything in its sister constituent. A priori, there is nothing stopping the sequence learner from thinking that "not" only modifies"the tall person." But the merge module knows that constituents should be considered as a whole, and that it is incorrect to let the individual meanings of the components escape a constituent.

2.1.3 Training Routine

To update its network, SAL solves a problem or a set of them. For each problem, it is solved when an operator returns whether it has been solved; otherwise SAL keeps applying operators. For each operator that SAL applies while training, it updates the gradients according to the procedure defined in this section.

It may always be possible to solve a given problem with a single composed operator, but the network is trained so that it has the ability to look ahead in the case that its operator choice actually does not solve a problem. I formalize the update procedure in a way that enables SAL to learn to solve problems by considering sequences of compositional operators if necessary.

The optimization procedure uses a policy network as well as a target network, as in Mnih et al.'s seminal neural reinforcement learning paper (Mnih et al., 2015). Every few state transitions, the target network is updated with the weights of the policy network. The loss, $\mathcal{L}$, is computed as in the below equations and updates are propagated through the network via the Adam optimizer (Kingma & Ba, 2014).

$$\delta = V^p(c) - (r + \gamma \max_{c' \in \mathcal{G}(p', o_p)} V^t(c')) \tag{2.1}$$

$$\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1 \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{2.2}$$

$$\mathcal{L} = \frac{1}{|B|} \sum_{(c,p',r) \in B} \mathcal{L}(\delta) \tag{2.3}$$

Equation 2.1

To the network, there is no a priori distinction between information about a problem and an action suggestion; they are handled together as one big mental condition input. Equation 2.1 represents the difference between the the policy network's evaluation of a mental condition, and the reward reaped from that mental condition combined with the target network's evaluation of the next mental condition. The next mental con-

dition contains the next problem state and goal and the operator from the generated ones that SAL thinks is best to take in this updated problem. Ideally, this difference should be zero. While there is a way to formalize this equation in terms of Q-values, I choose to view δ in terms of state values, because the network outputs only one value. Many such systems output a sequence of distinct values each for a sequence of discrete actions, as in Mnih et al.'s paper (Mnih et al., 2015).

V^p is the function given by the policy network, which takes in c, which is a mental condition. V^t is the same function as V^p, except it is given by the target network. r is the reward that is given upon the transition from c, after taking the compositional action described within c. And γ is the discount factor.

Here, I explain the maximization in Equation 2.1. c' is the mental condition that captures the updated problem's state and goal (denoted as p') after applying the compositional operator suggested in c. $\mathcal{G}(p', o_p)$ is as defined in Section 2.1.1. $\mathcal{G}(p', o_p) \subset \mathcal{C}(p', o_p)$, where $\mathcal{C}(p', o_p)$ is the set of all possible mental conditions; that is, the set of all such possible compositional operators paired with p'. This set could be infinite, so it is necessary to take a sample, $\mathcal{G}(p', o_p)$, from it. $\mathcal{G}(p', o_p)$ is regenerated each time this update step is taken. Note that, because $\mathcal{G}(p', o_p)$ is a set, the generator usually generates more than $|\mathcal{G}(p', o_p)|$ operators and finishes when $|\mathcal{G}(p', o_p)|$ unique operators have been generated. The critical assumption is given in Equation 2.4. The exactly correct learning of the estimation of mental condition values relies on this equation.

$$\max_{c' \in \mathcal{G}(p', o_p)} V^t(c') = \max_{c' \in \mathcal{C}(p', o_p)} V^t(c') \tag{2.4}$$

I do not believe that there is any reason that this assumption should hold in all cases, or that even the approximation should hold. The space of actions is not a continuum, and there could always be one more compositional nesting $\in \mathcal{C}(p', o_p)$ and $\notin \mathcal{G}(p', o_p)$ which would have completely changed whether a compositional operator actually works to solve a problem or not. However, if we assume that the world is sufficiently simple so that good progress can always be made on problems with

compositional operators of depth $< k << |\mathcal{G}(p', o_p)|$, and if the generator has a bias to generate less deeply nested operators, then I expect Equation 2.5 to hold.

$$P[\max_{c' \in \mathcal{G}(p', o_p)} V^t(c') = \max_{c' \in \mathcal{C}(p', o_p)} V^t(c')] \approx 1 \tag{2.5}$$

This result holds under my assumptions because the probability that the generator suggests all possible operators with a composition depth less than k approaches 1 as more operators are generated. So, the gradients will correctly update the network if $|\mathcal{G}(p', o_p)|$ is large enough.

Equation 2.2

$\mathcal{L}(\delta)$ in Equation 2.2 is the Huber (Huber, 1964) loss, which can minimize the magnitude of the value estimation error, δ, without producing unreasonably large gradient updates when δ is large.

Equation 2.3

In Equation 2.3, B is a batch of transitions from one mental condition to the next Innerese problem state and goal (this state and goal then has a compositional operator added to it as part of the maximization in Equation 2.1), with the corresponding reward. B is sampled from replay memory (Mnih et al., 2015) of such transitions that SAL experiences from exploring.

Exploration

Above, I provide the update equations that enable SAL to learn from choosing operators which return a reward signal. But how does SAL actually explore the space of operators and problem configurations in the first place? While it is important for $|\mathcal{G}(\cdot, \cdot)|$ to be very large to update the network in the correct way, the learner need not always choose what it considers to be the best operator (initially, SAL's weights will almost certainly be wrong, and they need to be updated). Although SAL always uses a large $|\mathcal{G}(\cdot, \cdot)|$ to perform the gradient updates, SAL learns by choosing

operators to try on a training problem, with a probability e, from a smaller subset of $\mathcal{C}(\cdot,\cdot)$ than $\mathcal{G}(\cdot,\cdot)$, which I call $\mathcal{G}(\cdot,\cdot)_e$. $|\mathcal{G}(\cdot,\cdot)_e|$ is set to be small enough so that SAL will be given sufficiently different operator choices upon each step (because $\mathcal{G}(\cdot,\cdot)_e$ is regenerated each time SAL selects an operator, along with $\mathcal{G}(\cdot,\cdot)$) which will provoke it to normally choose different operators upon each try. Yet, as long as $|\mathcal{G}(\cdot,\cdot)_e| > 1$, SAL's choices will never be fully random, because SAL will still pick what it thinks is the best operator in $|\mathcal{G}(\cdot,\cdot)_e|$. This feature enables SAL to try new operators during the learning process, with a bias towards operators that it already learned are good, and away from operators that it already learned are dangerous (e.g., perhaps it recognized early on that a particular operator sets the solution back by a step, but still needs to explore to learn the solution in its entirety). Note that, due to the use of $\mathcal{G}(\cdot,\cdot)_e$, SAL does not need any random exploration parameter that decreases as it learns (like many typical reinforcement learning systems (Mnih et al., 2015)).

In other words, SAL uses the generator to generate mental conditions from which one will be chosen and have its operator applied during learning; this results in state changes that are added to replay memory. And SAL also uses the generator as part of the gradient update process. It is important that the generator generates as many candidates as possible to correctly update the gradients. But, it is not crucial that the generator generate many candidates when SAL is learning and electing an operator to actually apply to a problem.

2.2 Self-Referential Operators

I posit that self-referential operators (at least, the type that operate at the natural and inner language level of cognition) are an important aspect of self-aware thought, and can assist the learner. I present five operator predicates that can be instantiated to form self-referential operators in this section, which are simple and easy to implement in the Innerese and Self-Aware Problem Solver framework, yet extend SAL's capabilities drastically and imply a variety of humanlike behaviors. Note that all of the predicates in this section are instantiated by the generator to create operators

which are chosen by the learner, just like any other predicate that SAL has access to. There is nothing special about them, in this sense.

Among other theoretical benefits, I believe that the real practical benefit that these operator predicates give SAL is an ability to learn to receive help from external agents, such as a human. Even though a human never provided SAL with a full solution, and in many cases the human input did not help at all, it is doubtful that SAL would learn successfully in my experiments without asking a human for help. But SAL could not reasonably be expected to acquire this help without some amount of self-reference. Without the following self-referential operator predicates, SAL would not be able to explain its problem to others, ask for helpful information to internalize as Innerese (and construct and carry out an operator from the Innerese's scope information), introspect about its own problem solving indefinitely, or remember and articulate its past actions. And SAL would not be able to learn how to use these capabilities in their own right or learn how to use them to help problem solving in the first place.

To view an example of the following predicates in action, refer to Figure 2-4, but ignore the information given by the braces at this time. To view their Innerese templates, refer to Figure 3-1. Rewards that they return will be given in Chapter 3.

Externalize converts the current Innerese problem state and goal, including any self-referential information about SAL's own problem solving that was added to the state, to natural language and prints it to the screen.

Internalize waits for natural language input from an agent such as a human. The natural language input is then converted to Innerese. If the input expresses an operator, then SAL carries it out; otherwise it fails to modify the problem. It is unlikely that **Internalize** will be useful if SAL does not first use **Externalize**, so that it can share information about the problem with the helper. **Internalize** also causes an unpredictable result. SAL never knows exactly what a helper will suggest or whether the help will actually be useful. Because SAL can learn how to apply instantiations of this operator predicate, this case is an example of how SAL can still learn how to use operators that have a probabilistic result. Ultimately, **Internalize** can help SAL circumvent the difficulty of learning without much a priori guidance

from the network, in a vast compositional action space.

Remember(x) takes in an operator as an argument, and inputs into the mental condition an Innerese memory of that operator, if there is not already a memory of it and if SAL has tried that operator before (even if that operator is a constituent of a larger operator, as long as it has actually been executed). In the future, this operator predicate could be extended to insert useful commonsense knowledge into the current mental condition, so that an underspecified problem could be solved in the first place. To be clear, SAL stores all operators tried before in a list, but in order to actually include a tried operator in the mental condition, **Remember(x)** must be used.

Forget(x) takes in a compositional operator as an argument, and if there is an Innerese memory of it in the current mental condition, then it is removed. It is a way to undo **Remember(x)**, as irrelevant memories of every operator SAL tries could otherwise overtake the length of the original mental condition, and make the learning problem harder. On top of this, externalizing all operators tried to a helper is usually unnecessary.

Then(x, y) enables the learner to solve problems that normally require multiple steps, with a single operator. It takes two compositional operators as arguments, and executes them in sequence.

2.2.1 Infinite Introspection from Finite Means

Some of these self-referential operators are recursive, meaning that they can introspect as much as memory limitations allow for; particularly, instantiations of **Remember(x)**, **Forget(x)**, and **Then(x, y)**. This is a seemingly human property, as a person can think about their own thinking, and then think about their own thinking of their own thinking and so on. For example, SAL can remember whether it remembered remembering something. SAL could also remember forgetting something or forget remembering something. And SAL may decide to externalize and then internalize and then remember this act so that it does not ask for help indefinitely. SAL might then remember the operators it takes after receiving help, and if it takes too many, externalize and then forget all of the previous operators and previous help

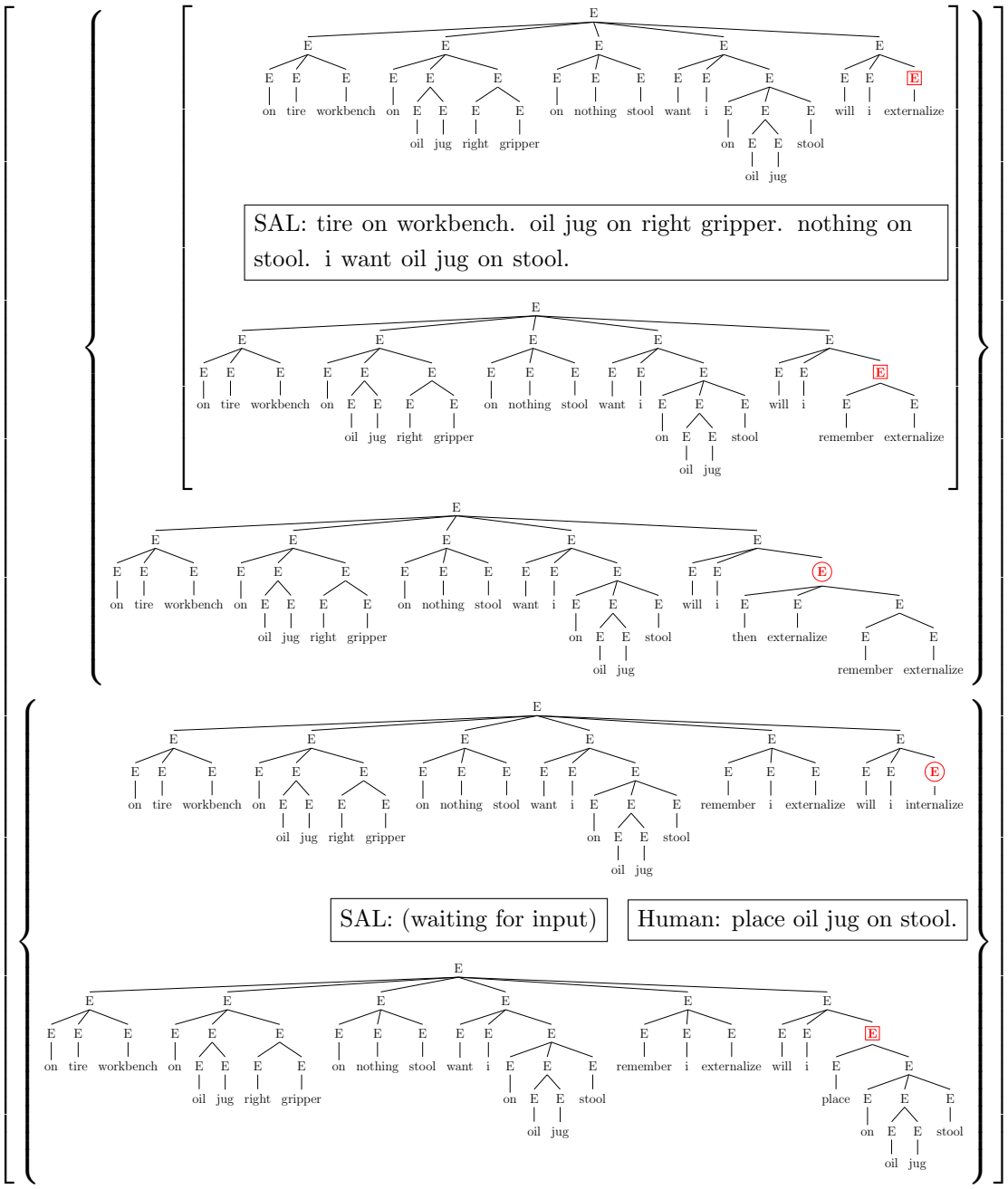

Figure 2-4: Examples of the self-referential operators. (The only operator predicate not used here is **Forget(x)**, but it could remove "(remember i externalize)" in the figure.) The corresponding English outputs and inputs are also provided. To the gradient update mechanism, mental conditions in square braces happen sequentially, and those in curly braces happen at the same time. Here, SAL chooses two operators (the roots of their Innerese descriptions are circled in red). The rest of them (rectangled in red) are forced from those chosen. For example, choosing "(then externalize (remember externalize))" causes SAL to also add to replay memory the transition from "externalize" followed by that from "(remember externalize)." The same is true about the temporary duality between "internalize," and the resulting human help, parsed into Innerese as: "(place (on (oil jug) stool))."

and then internalize and then remember this act.

2.2.2 Provoking New Ways to Learn

Some of SAL's self-referential operators evoke sub-problems that could require multiple steps; particularly instantiations of **Then(x,y)** and **Internalize**. They can also do this recursively (e.g., a human's input from **Internalize** can be a use of a **Then(x,y)** instantiation). When SAL internalizes instructions for how to make progress on a problem, it makes a temporary goal to take all of the operators that were suggested to it. **Then(x,y)** also makes a temporary goal to try a sequence of operators.

In a sense, these self-referential operators lead SAL to take multiple actions at the same time. Suppose that the operators that are used within the sub-problem from **Then(x,y)** or **Externalize** lead to the following transitions.

$$(c_1, p'_1, r_1)...(c_n, p'_n, r_n) \tag{2.6}$$

These transitions are added to memory replay for backpropagation as normal. However, the following transition is also added.

$$(c_{1'}, p'_n, r_n) \tag{2.7}$$

$c_{1'}$ contains the same state and goal Innerese as c_1, but the operator that is expressed in $c_{1'}$ is the self-referential operator that provokes the sub-problem, and not the first operator in the sequence that comprises the sub-problem. From one perspective, SAL solved the whole sub-problem in one step with a self-referential operator; but from another perspective, SAL took, potentially several, operators to solve the sub-problem, without the self-referential operator. And SAL can learn both of these perspectives to understand, for example, when it is useful to ask for help in general, and when it is useful to apply an operator that was recieved with help.

To view an example of this duality between self-referential operators and operators that they induce, refer to Figure 2-4, and give attention to the the information about

the braces.

2.2.3 Explaining to Oneself and Overcoming Partial Observation

It is clearly useful for SAL to be able to explain the problem to ask for help (sometimes listing actions it might have taken to get to its current state via **Remember(x)**, or intentionally not including actions that it considers not useful to the explanation via **Forget(x)**). But it is also essential for SAL to develop explanations that need not be externalized to others. Explaining what you did to yourself is a way to overcome Partially Observable Markov Decision Processes (POMDPs). For example, SAL actually faces a POMDP when it wants to **Internalize** help from others, because unless it explains its problem via **Externalize**, then others may not be able to help. Under the current architecture, there is not any automatic update to the mental condition that says whether SAL explained itself already or not; in this case SAL must use **Remember(x)** to receive this information. As a side note, SAL would still need to consider self-referential information even if **Externalize** automatically inserted Innerese information about its use (because this inserted information would be self-referential).

The notion of recurrence to handle POMDPs is not new, but the recurrence is typically built into the value-selecting network itself (Hausknecht & Stone, 2015). This prevents a typical system from easily being able to articulate in natural language a scenario such as "I remembered that I just explained my problem to someone. If I did not remember that action, then I would have chosen to explain my problem to someone again. Because I remembered that action, I now choose to wait for natural language help from them." SAL is capable of actually articulating what it remembered.

Chapter 3

Experiments

The goal of the following tests is not to benchmark SAL against competitors on any particular dataset. There are not any competing systems which I am aware of, and I view the data that I provide to be mainly of theoretical interest. I also do not empirically test all of my thought experiments from the previous chapter. The goal is instead to provide evidence that my system meets my most fundamental claims about the humanlike faculties that emerge from SAL. I first demonstrate that SAL works; that is, SAL can learn to solve a nontrivial multi-step problem, and that it could not have done so (at least, with the provided training data) without using the provided self-referential operators, or some other form of self-reference, to receive help from a human. My next demonstration is that SAL, like any useful learning system, can generalize. SAL can generalize to new problems, goals, and operators, by bootstrapping with its existing knowledge, even if it has seen only one training problem. These experiments provide some understanding of how and why humans have the capacity for the kind of (presumably self-aware) thought that can be translated to and from natural language. The experiments also highlight practical benefits of SAL, which could be implemented in production question-answering systems.

In all of the following tests, unless stated otherwise, the hyperparameters were: $r = 1$, if the corresponding operator returns that a problem is solved, and $r = -0.1$ otherwise, $|\mathcal{G}(\cdot,\cdot)| = 500$, $|\mathcal{G}(\cdot,\cdot)_e| = 2$, $e = 2/3$, LSTM hidden state size = 100, $|B| = 20$, target update = every 10 state transitions, replay memory size = 200,

$\gamma = 0.99$, Adam learning rate = 0.0003. The pre-trained word embeddings used were 100 dimensional GloVe embeddings (Pennington et al., 2014). Finally, to prevent the time-consuming process of generating lengthy compositional operators (usually caused by the binary branching tendency of **Then(x,y)**), the generator was limited during training and evaluation to only generate operators with a total of eight arguments or fewer.

3.1 Learning Task

It is time to revisit the manipulation domain and simulator from Section 1.4. This domain is implemented within a simulator that eliminates the need for perception capabilities that translate the state of the world into Innerese; building such perception modules is outside of the scope of this thesis. In addition to the self-referential operator predicates described in Section 2.2, SAL is given **Pick(x)** and **Place(x,y)**, where should be x is some movable object in the world and y should be some object in the world. It is possible for SAL to instantiate and try these predicates with the wrong types of Innerese constituents, but they will simply fail to do anything, because the simulator is programmed to know the allowable types of objects. In order to use **Place(x,y)** successfully, x must first be either on the left or right gripper of SAL's virtual body; x can only get to these locations with **Pick(x)**. **Pick(x)** first tries to transfer x to the left gripper. If there is an item already on the left gripper, then it tries the right gripper; if both grippers have items on them then **Pick(x)** fails to change the state of the problem. In addition, **Pick(x)** will not change the state of the problem if there is another object on top of x. Again, while this knowledge is encoded in the simulator, SAL must learn it.

Pick(x) and **Place(x,y)**, after the end of their application, update the Innerese and check the state to return whether the problem was solved ($r = 1$ or -0.1). Because **Internalize** and **Then(x,y)**, evoke a sub-problem, they return whether the last operator returns if it solved the problem ($r = 1$ or -0.1); more detail is given about these operators in Section 2.2. All other operators do not do any checking and

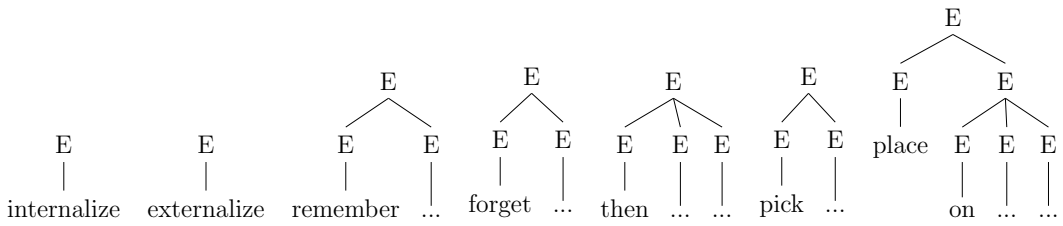

Figure 3-1: The actual Innerese representation of the operator predicates that I have introduced.

so always return that the problem is not solved ($r = -0.1$).

The actual Innerese representations of the operator predicates that SAL has access to are shown in Figure 3-1.

3.1.1 Specific Problem

For my tests, the world in which SAL learns has two movable objects (denoted by "tire" and "oil jug") and two immovable objects (one denoted by "stool" and one denoted by "workbench"). Initially, the oil jug is on the workbench and the tire is on the stool. The goal is for the oil jug to be on the stool. This goal requires a minimum of two steps; one step is required to pick the oil jug up, and another is required to put it down on the stool.

3.1.2 Rationale

I consider the difficulty of this domain, and the specific training problem, to be nontrivial. The **Pick(x)** and **Place(x,y)** operators require SAL to learn the correct arguments, just as humans must. In the specific problem that SAL learns, SAL must also either learn to use a compositional operator beyond a single **Pick(x)** or **Place(x,y)**, or learn to take multiple steps to solve the problem. If the agent learns a successful multiple-step policy, then there is good evidence that my gradient update equations work as anticipated, because SAL must take a first step that returns a negative reward. Also, the training problem requires that SAL learn what not to do: SAL can create a harder problem for itself by accidentally stacking an object on top

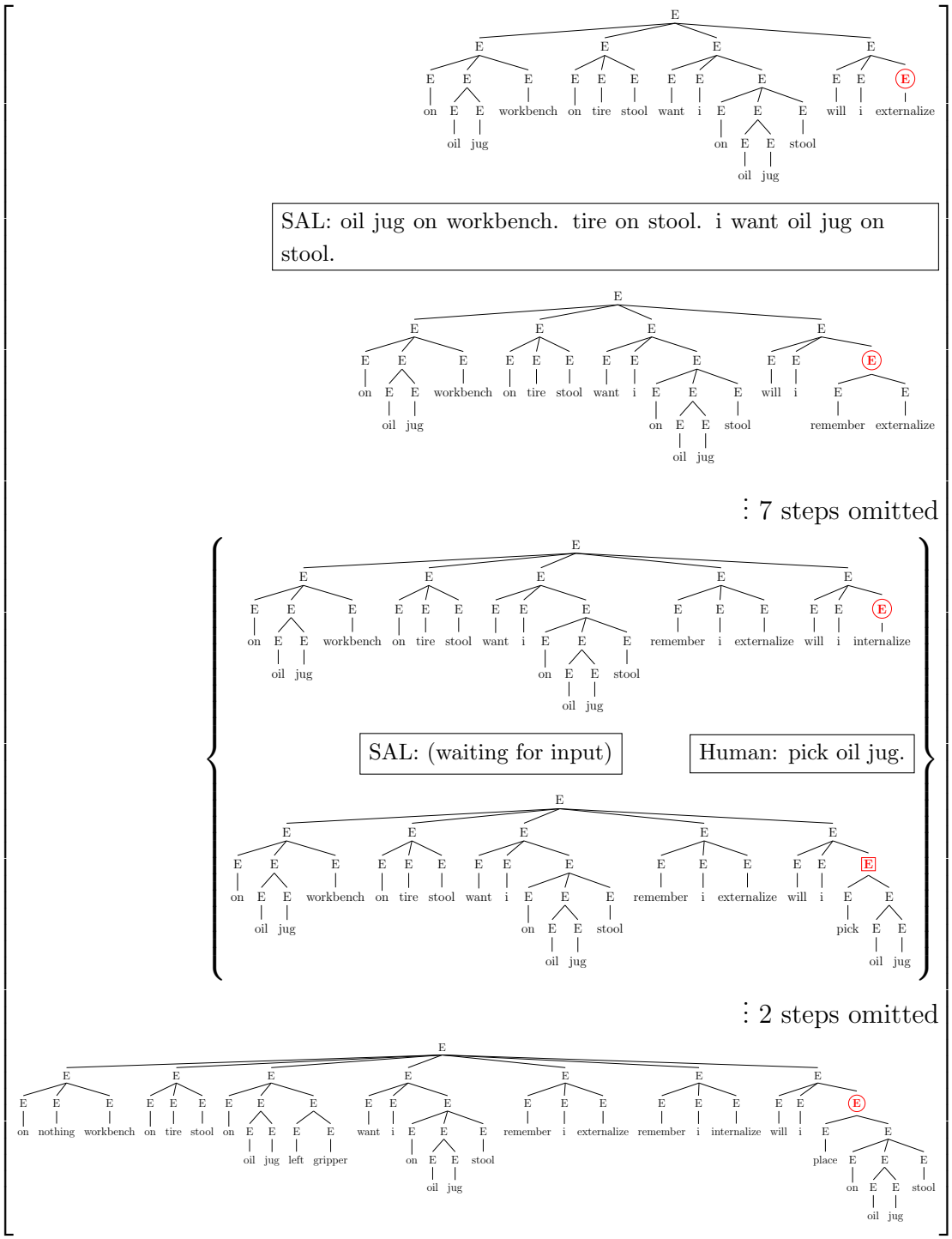

Figure 3-2: A mental condition sequence that SAL had, on the 38th iteration of training, before solving the problem. The corresponding English outputs and inputs are also provided. To the gradient update mechanism, mental conditions in square braces happen sequentially, and those in curly braces happen at the same time. The roots of chosen operators' Innerese descriptions are circled in red. The roots of those rectangled in red are forced from those chosen.

of another object that must be moved, and SAL must learn near-miss differences, (Winston, 1970) such as "(place (on (oil jug) stool))" versus "(place (on stool (oil jug)))" or "(place (on oil stool))." Finally, the problem is complicated enough that learning even a single step is challenging; there are infinite possible operator compositions, from which a helpful one must be elected. Even if operator construction is limited such that they must contain eight arguments or fewer, the number of possible compositional arguments to choose from is in the tens of thousands. Even among just the possible **Pick(x)** and **Place(x,y)** instantiations, there are about 200 possible operators to choose from, and only one (namely, "(pick (oil jug))") will advance the solution. I am referring to the branching factor: if SAL proceeds to pick up the oil jug, then there is again a large number of compositions to choose from to finish solving the problem. I cannot give an exact number for the branching factor at all steps because it changes depending on the number of Innerese constituents that the generator has access to, which changes depending on the problem's current state and goal.

3.2 SAL Learns an Optimal Solution

SAL learns by solving the training problem 40 times. This is a hard-coded hyperparameter; SAL must apply operators to find the solution, and then the problem starts over, 40 times. As a reminder, SAL explores by applying operators, with guidance from its network that improves over time (outlined in 2.1.3). Every time SAL asks for input via **Internalize** and it has already used **Externalize** to tell me the problem, I provide it with a uniformly random choice from the following two (usually, but not always, helpful) natural language productions: "pick oil jug" and "place oil jug on stool." I never gave SAL the whole solution at once. If SAL does not externalize the current problem by the time it uses **Internalize**, then I provide it with an empty string to internalize, which essentially wastes that move by telling SAL that **Internalize** failed. A real sequence of SAL's mental conditions in this world from the 38th epoch of training, is shown in Figure 3-2.

After training, I set $|\mathcal{G}(\cdot,\cdot)|$ and $|\mathcal{G}(\cdot,\cdot)_e|$ to be 1000 and disabled gradient updates. These changes to the set sizes increase the likelihood that SAL will choose, what it considers to be, the best operator possible, while still being small enough so that the generator does not generate bizarre compositions that SAL has never seen before. Upon allowing SAL to perform on the problem, it chose the following sequence and solved the problem: "(pick (oil jug)), (place (on (oil jug) stool))."

At the end of the 40 epochs, SAL learned a two-step solution with the optimal number of item movements. It actually solved the problem with these optimal moves (and without any help) as early as the 27th iteration; but I allowed SAL to complete the predetermined number of training epochs because early stopping for evaluation purposes equates to p-value hacking. While SAL learned, it first learned to rank both **Internalize** and **Externalize** highly so that it could receive help. After gaining more experience, a tendency emerged for it to ask for help less often. The entire action sequence that SAL used during training can be found in Table A.1. Another interesting point is that SAL seems to ask for help via **Internalize** more often as the test problem deviates more from the training problem, as in Table 3.2, which will soon be discussed. Although, in many of my evaluations, SAL does not show evidence of knowing to first **Externalize** and then **Internalize**.

3.3 SAL Cannot Learn Without Self-Reference

I tried training SAL without any of the self-referential operators. Training proceeded just as before, except SAL only had access to **Pick(x,y)** and **Place(x)**, and $|\mathcal{G}(\cdot,\cdot)|$ was set to be 150. I could not keep $|\mathcal{G}(\cdot,\cdot)|$ at 500, because during most steps, there were about 200 possible operators to choose from. The proportional size of $|\mathcal{G}(\cdot,\cdot)|$ to the number of possible operators was actually far larger in the non-self-referential case. It took 41191 pick and place operators to complete the training routine, compared to the 344 such operators (90 of which were from instances of natural language input) that SAL, with access to the self-referential operators, took. And after training without the self-referential operators, SAL was not able to learn to

solve the problem, or even get closer to a solution. Upon also switching $|\mathcal{G}(\cdot,\cdot)_e|$ to 150 for evaluation, SAL's attempted solution was completely random, which exceedingly rarely even contains a production that I consider well-formed, let alone a production that actually contributes to a solution. Upon further inspection, SAL trained without the self-referential operators ranked every generated mental condition as having a value of -9.8959, which explains why its behavior was completely random. The reward signals were so sparse that SAL literally could not learn to distinguish any mental condition generated.

It is apparent that SAL needs some natural language help from a human to find the useful operators in the haystack that is the vast compositional action space. And SAL needs all of its self-referential operator predicates to do so. It needs **Then(x,y)**, **Remember(x)**, and **Forget(x)** to tackle the partial observality involved in externalizing and then knowing to internalize, and it needs **Externalize** and **Internalize** in their own right. Even though the self-referential operator predicates add infinite possible actions instead of the, roughly 200 operators that the non-self-referential operator SAL had access to, the help from the self-referential operators still enabled SAL to learn with the given amount of data. I have no doubt that SAL would eventually be able to learn a good policy without self-referential operators, but it appears that this will require significantly larger batch sizes and/or more epochs.

Even if SAL does not choose to use self aware operators during performance, they are still immensely helpful during the learning process. Also, as I demonstrate in the next section, the use of self-referential operators does not result in a "hard-coded" network that cannot generalize to new tasks.

3.4 SAL Generalizes by Bootstrapping from Syntactic Constraints and Semantic Information

In this section, I demonstrate SAL's ability to generalize to previously unseen problems, after SAL is trained with access to the self-referential operators on my training

problem from the previous section. The new problems contain new words in the place of those in the training problem. I also conduct an experiment where new words replace those in the Innerese templates for some of SAL's existing operator predicates. In these tests, SAL must demonstrate some form of bootstrapping to know the operational meanings of the new elements that I introduce. I believe that SAL could also generalize to solve problems involving completely different arrangements of movable and immovable objects than those that it has seen before, but it is unreasonable to expect this performance after learning from just one problem. (SAL can already solve the training problem when the tire and oil jug are in a different initial configuration, but this is expected, because SAL moved the tire and oil jug into many configurations before it solved the problem on many training epochs, and SAL needs to put the state of the problem into an intermediary configuration before solving it anyway.)

SAL has an inductive bias that enables it to guess the operational meanings of new words based on their given Innerese syntactic constraints, and word embeddings, with no additional training. After training, I replaced the Innerese words in the templates for **Pick(x)** and **Place(x,y)** with a variety of words seen and discussed in Table 3.1. I then evaluated SAL in the same way that I evaluated it on the training problem previously. Even though SAL had never seen these words before, it knew the templates and that their corresponding operator predicates took the same number and type (Innerese constituent or operator) of arguments as those of **Pick(x)** and **Place(x,y)**. SAL also knew the word embeddings of the new words. This information was enough for SAL to perform, in many cases, at the same level as if "pick" and "place" had been used. Presumably SAL is able to do this by, existing in a certian state with a certian goal and using the generator to generate instantiations of an operator predicate. For example, SAL might then notice that, due to the syntax and embedding of the operator predicate represented by a new word called "set", "(set (on (oil jug) stool))" is a possible construction, and remember from training that "(place (on (oil jug) stool))" was an Innerese representation for a valuable action to take in its current state. Then SAL would use "set" along with its desirable arguments. Because SAL can also achieve such

performance with embeddings for words that are not semantically related to "pick" or "place," such as "scrophularia," Table 3.1 shows evidence that SAL can bootstrap this knowledge from syntactic information alone. Note that treating "scrophularia" as similar to "pick" is normally a silly thing to do. But I believe that it is a sensible choice when SAL doesn't know the operational meaning of "scrophularia," yet does know that its operator has the same syntax as the "pick" operator. In this case, it has no knowledge to go by except the syntax, and a word embedding that is not similar to any embeddings it has seen before. This capability can be seen as a prototypical version of syntactic bootstrapping (Gleitman, 1990). Humans use the syntax of words to infer meaning, and SAL can do the same by using the syntax of a new word that stands for an operator predicate to decide upon a correct use for it. However, SAL typically performs at a higher level if the replacement words are semantically related.

I also replaced the word "oil" in the training problem's state and goal with a variety of semantically related and unrelated alternatives, and evaluated SAL on it to determine whether it could infer the correct operational use of the replacement word. I found that, among both semantically related and unrelated words, SAL indeed had a useful policy, even though semantic relationship to "oil" helped significantly. SAL had a useful policy with semantically unrelated words by virtue of information other than embedding knowledge. For example, the representation of the problem and goal includes the constituent "(compilers jug)" when the replacement word is "compilers." This syntactic association with "jug" explains how SAL is still able to achieve the correct solution by filling **Pick(x)** and **Place(x,y)**'s x arguments with the correct item, even though the item may contain a practically nonce word. But, because I was able to show a significant performance difference between semantically related versus unrelated words, I have provided evidence that SAL can bootstrap useful operational meaning from word embeddings. One can possibly think of this capability as a prototypical version of semantic bootstrapping (Pinker, 1984), which is the human capability to infer syntactic categories based on semantic information about a word. My association of this behavior with actual semantic bootstrapping is not central to this thesis; it is simply my interpretation of this aspect of SAL's ability

"pick, place" Swaps	Attempted Solution
grab, set	**(grab (oil jug)), (set (on (oil jug) stool))**
pluck, rest	(pluck (oil jug)), (pluck (on (oil jug) stool))
grip, put	(grip (oil jug)), (grip (on (oil jug) stool))
get, leave	**(get (oil jug)), (leave (on (oil jug) stool))**
acquire, deposit	(acquire (oil jug)), (acquire (on (oil jug) stool))
grasp, lay	**(grasp (oil jug)), (lay (on (oil jug) stool))**
clasp, position	**(clasp (oil jug)), (position (on (oil jug) stool))**
clutch, situate	**(clutch (oil jug)), (situate (on (oil jug) stool))**
grapple, settle	**(grapple (oil jug)), (settle (on (oil jug) stool))**
take, sit	(take (oil jug)), (take (on (oil jug) stool))
guiltiest, phishers	(guiltiest (oil jug)), (guiltiest (on (oil jug) stool))
98-93, kranhold	(98-93 (oil jug)), (then (kranhold (on (want i (on (oil jug) stool)) stool)) (98-93 stool))
swineflu, fanck	(swineflu (on (oil jug) stool)), (swineflu (on (oil jug) stool))
toner, assizes	**(toner (oil jug)), (assizes (on (oil jug) stool))**
titanosaur, märjamaa	(titanosaur (oil jug)), (märjamaa (on (on (oil jug) stool) stool))
archeparchy, grella	(archeparchy (oil jug)), (then (archeparchy stool) (archeparchy jug))
bacteroidetes, cowered	(bacteroidetes (on (oil jug) stool)), (bacteroidetes (on (oil jug) stool))
maritza, stylinski	(maritza (oil jug)), (maritza (on (oil jug) stool))
siniora, maurycy	(siniora (oil jug)), (then (siniora stool) (maurycy (on tire stool)))
scrophularia, attests	**(scrophularia (oil jug)), (attests (on (oil jug) stool))**

Table 3.1: Evidence for prototypical syntactic and dual prototypical syntactic-semantic bootstrapping. The table shows SAL's performance when the words "pick" and "place" are replaced in SAL's Innerese description of the **Pick(x)** and **Place(x,y)** operator predicates by those in the table. SAL's solutions were cut off after two moves. Ten of the word pairs were chosen to have meanings that I consider to be, roughly synonymous with "pick" and "place." The other ten of the word pairs were selected from the GloVe embedding dictionary uniformly at random. The bold words are those that result in the optimal solution. SAL's behavior is highly significant; the data shows that SAL can generalize to use new operator words by virtue of both the provided syntax and the word embeddings. Among the cases with semantically unrelated words where it is reasonable to expect SAL to rely on the operator predicate syntax alone, SAL is able to advance the solution ($p < 1.940e\text{-}16$ with the null hypothesis that SAL chooses operators randomly). And SAL was even able to solve the problem fully in a couple of these cases. Yet, dual word embedding information paired with the syntax also appears to be more beneficial than one type of information alone ($p < 8.087e\text{-}37$). There is some, yet insignificant, evidence that SAL's performance with the semantically related word embeddings is better than that without ($p=0.140$ with James's nonparametric test (1954), when the first step is the first dimension the second step is the second dimension; first and second steps have the same replacement word, so they cannot be assumed to be completely separate tests).

to generalize. This is evidenced and discussed in Table 3.2.

"oil" Swaps	Attempted Solution
petroleum	**(pick (petroleum jug)), (place (on (petroleum jug) stool))**
petrol	**(pick (petrol jug)), (place (on (petrol jug) stool))**
gasoline	**(pick (gasoline jug)), (place (on (gasoline jug) stool))**
fuel	**(pick (fuel jug)), (place (on (fuel jug) stool))**
lubricant	**(pick (lubricant jug)), (place (on (lubricant jug) stool))**
grease	**(pick (grease jug)), (place (on (grease jug) stool))**
lubrication	**(pick (lubrication jug)), (place (on (lubrication jug) stool))**
kerosene	**(pick (kerosene jug)), (place (on (kerosene jug) stool))**
diesel	**(pick (diesel jug)), (place (on (diesel jug) stool))**
napalm	**(pick (napalm jug)), (place (on (napalm jug) stool))**
ivic	(pick (ivic jug)), internalize
showed	(then (pick stool) (remember (pick workbench))), (then (pick (on tire stool)) (pick jug))
murigande	(pick (murigande jug)), internalize
chelios	**(pick (chelios jug)), (place (on (chelios jug) stool))**
aricie	(pick (aricie jug)), internalize
ligule	**(pick (ligule jug)), (place (on (ligule jug) stool))**
oom	**(pick (oom jug)), (place (on (oom jug) stool))**
fedorchenko	(pick (fedorchenko jug)), internalize
haugesund	**(pick (haugesund jug)), (place (on (haugesund jug) stool))**
compilers	**(pick (compilers jug)), (place (on (compilers jug) stool))**

Table 3.2: Evidence for prototypical semantic, syntactic, and dual prototypical syntactic-semantic bootstrapping. This table shows SAL's performance when the word "oil" is replaced everywhere that it occurs by another word. Ten replacement words were chosen due to their semantic similarity to "oil" and ten words were chosen uniformly at random from the GloVe embedding dictionary. There is a significant difference in SAL's performance when the word is semantically similar to "oil" (p=0.0319 with James's nonparametric test (1954), applied analogously to that in Table 3.1). This is evidence that SAL's word embedding knowledge can yield significant performance increases. Also, SAL can make use of both syntactic and semantic information to bootstrap a significant understanding in the cases with words that have semantic similarity to "oil" ($p < 9.537e\text{-}47$, again, under the null hypothesis of random operator choices). On top of this, the data also provide evidence, in addition to that in Table 3.1, that SAL can use syntax alone to perform significantly well in the presence of practically nonce word embeddings ($p < 2.246e\text{-}28$).

Chapter 4

Discussion

SAL, to the best of my knowledge, is the first system that can learn to improve at using an aspect of, what I consider to be, self-awareness; particularly the kind that is necessary to externalize and internalize natural language about one's own problem solving. And I also believe that SAL is the first system to use this capability to learn how to improve at problem solving. In this chapter, I consider the future steps for this work, and I also present my existing technical contributions.

4.1 Next Steps, Scaling Up, and Practical Use

I provided evidence that self-referential operators can take a problem solver's learning capabilities to another level. And I presented some self-referential operator predicates that demonstrate this point. But in the future, SAL's existing self-referential operator predicates could be refined, new ones could be added, or they could be replaced entirely by more primitive ones that simply have access to SAL's mental condition history, and can read, write, and externalize any word or character. An obvious practical benefit of SAL's self-referential operators, is that SAL can explain itself to a human and receive help. But this idea can be extended so that multiple SAL instances can share knowledge by externalizing and receiving help from each-other in natural language, enabling a sort of parallel, crowd-sourced, learning.

Even though SAL can learn to select a good operator when there are hardly any

useful ones in a large collection of candidates, actually generating and evaluating mental conditions with so many operators may become computationally intractable for large problems. One way to overcome this could be keeping a set of operators found to be useful during training, and always (or almost always) adding them to the set of generated operators, instead of waiting until they are generated by a random procedure. Another way to overcome this problem is to add a neural generator that learns to generate useful compositions (almost an inverse of SAL's current learning mechanism, which learns to evaluate generated compositions). There may be room to use an approach similar to a Generative Adversarial Network (Goodfellow et al., 2014) for this purpose: a generator that suggests useful actions and a discriminator that tries to find a difference between the suggested action and some optimal move.

SAL assumes that the argument structure of operator predicates is given, but the only mechanism that uses the argument structure is the generator. It is a simple modification to create a generator that does not require the argument structure and instead generates operators with many different possible arguments and numbers of them (they would just fail to run if the argument structure is not correct). I only require the argument structure to be given in my thesis, so that the learning process can be simplified and I can focus on the main ideas behind SAL. SAL would presumably still be able to do some form of syntactic bootstrapping if told about the argument structure of new operator predicates, possibly through example.

Another source of SAL's knowledge is within the word embeddings, but neither the embeddings nor the exact dictionary of words need to be fixed, as in my thesis. I used the GloVe embeddings to give SAL a priori knowledge about what words might be similar operationally. It is conceivably possible for SAL to learn the operational meaning of a word that is not even in its embeddings, through experience and/or any syntax that is given, and actually construct an embedding for it in the process. All that is required is for gradient updates to happen in a newly initialized word embedding in addition to those that already happen in the merge module. The gradient updates to a new word embedding may even push it towards similarity with existing word embeddings that represent items that SAL already knows about and

can use in similar ways. SAL could also modify existing word embeddings through this process.

Finally, SAL also assumes a parser and translator that can produce logical forms of statements from natural language and can translate logical forms back to natural language. But this can be thrown away. A small modification would give SAL the ability to learn an inner language structure of natural language strings by itself, while still not precluding the use of being told the structure explicitly. None of the ideas behind the SAL architecture rely on a specific type of hierarchical parse, such as Innerese. SAL could aggregate the results of the merge module on all possible hierarchical parses at once. This would require more computational power, as there would be far more neural merge operations and far more constituents in the generator procedure. The hierarchical parse constituents would not be given; SAL would have to use all possible parse constituents to construct compositional operators. But I do not believe that the computational burden will prove to be unfeasible. The merge module already uses a memoization of embeddings of existing constituents so that it does not re-merge them when computing embeddings in several related batches of many problem states, goals, and operators. I consider this paragraph to contain the most important idea in this section.

With these modifications, I believe that SAL will be able to scale to large production problem-solving and question-answering status. I also believe that it will be able to do so with a reasonable amount of training by bootstrapping existing knowledge as I have already demonstrated.

4.2 Contributions

I end this thesis by summarizing my technical contributions. I found evidence that a variety of humanlike behaviors emerge from SAL, including the ability to generalize to new problems and new representations for operators by virtue of lexical and syntactic patterns. I also provided evidence that SAL could not have reasonably learned the training problem without the capabilities given by the self-referential operators

and without a representation that is translatable to and from natural language. Pivotal capabilities are explaining oneself (either to oneself or others) and receiving help. On top of this, my thesis can be viewed as presenting a way to extract operational knowledge from word embeddings, and so is part of a line of research that investigates the knowledge inherent in word embeddings. In order to accomplish these, feats, I contributed a new neural learning apparatus that follows hierarchical syntactic constraints present in theories of human language, and I contributed a new loss function to train it. Ultimately, I have introduced a new way to do reinforcement learning, using a system that considers compositional states, goals, and actions, as one. Because of the generality of my system and its ability to use self-referential operators to solve otherwise intractable problems given the training data, my thesis can also be seen as bringing reinforcement learning a step closer to working in the real world instead of a simulation. To the best of my knowledge, typical neural reinforcement learning systems are only successful in simulations where a large amount of training data can be generated in a reasonable amount of time. I believe that all of my results indicate a serious hope of scalability. Finally, due to everything that I have shown, I have provided a rationalization for how and why humans may have aspects of self-aware thought, such as the ability to recursively share with oneself and others, and internalize, helpful information that references one's own problem solving.

Appendix A

Operator Choices from Training

Table A.1: SAL's operator choices during training. Each of the 40 epochs through the training problem is separated by a line. From the perspective of the gradient update procedure, operators within square braces happen sequentially and operators within curly braces happen at the same time. Only one of the operators in a collection designated by curly braces is actually chosen by SAL, and the rest are forced. Due to this, it may appear at first glance that SAL chooses more operators than it actually does. Also note that the generator is limited during training for SAL to explore the space of possible operators; SAL's choices during training do not necessarily reflect the operator that it thinks is best out of all others. Refer to the text for more detail.

(pick (on tire stool)), { [(pick (want i (on (oil jug) stool))), (remember internalize)], (then (pick (want i (on (oil jug) stool))) (remember internalize)) }, (place (on i (oil jug))), (place (on oil (oil jug))), externalize, (forget externalize), { [(pick tire), (place (on (oil jug) ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool)))))], (then (pick tire) (place (on (oil jug) ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool)))))) }, (remember externalize), externalize, { internalize, (pick (oil jug)) }, externalize, (place (on tire workbench)), { [externalize, (place (on jug gripper))], (then externalize (place (on jug gripper))) }, externalize, { internalize, (place (on (oil jug) stool)) }
(remember (place (on (on (oil jug) workbench) stool))), externalize, { internalize, (pick (oil jug)) }, (forget (place (on on (left gripper)))), { internalize, (pick (oil jug)) }, (place (on (right gripper) gripper)), externalize, (forget internalize), (remember (pick (on (oil jug) stool))), externalize, externalize, (place (on nothing on)), { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
(forget (remember (place (on on (on (oil jug) workbench))))), internalize, internalize, internalize, externalize, externalize, (forget (pick ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))))), { internalize, (place (on (oil jug) stool)) }, { internalize, (pick (oil jug)) }, externalize, externalize, { [{ internalize, (place (on (oil jug) stool)) }, { internalize, (pick (oil jug)) }], (then { internalize, (place (on (oil jug) stool)) } { internalize, (pick (oil jug)) }) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
(place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) (on (oil jug) workbench))), internalize, internalize, (place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) (on (oil jug) stool))), internalize, internalize, externalize, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
(pick tire), (pick want), (place (on nothing (on nothing stool))), (place (on tire gripper)), internalize, (remember externalize), { [(remember internalize), internalize], (then (remember internalize) internalize) }, internalize, internalize, (forget (remember internalize)), (remember (forget internalize)), (remember internalize), externalize, externalize, { internalize, (place (on (oil jug) stool)) }, externalize, { internalize, (place (on (oil jug) stool)) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
externalize, (place (on workbench (on tire stool))), (place (on (on tire stool) (want i (on (oil jug) stool)))), (remember (forget externalize)), externalize, { internalize, (pick (oil jug)) }, externalize, (pick workbench), { internalize, (place (on (oil jug) stool)) }

(pick i), externalize, { internalize, (pick (oil jug)) }, externalize, externalize, externalize, { internalize, (place (on (oil jug) stool)) }

---

internalize, { [(place (on (oil jug) (on (oil jug) stool))), (forget (remember (pick tire)))], (then (place (on (oil jug) (on (oil jug) stool))) (forget (remember (pick tire)))) }, internalize, externalize, (pick on), (place (on (on (oil jug) stool) (on tire stool))), (place (on (oil jug) (oil jug))), { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, (forget externalize), (pick stool), { internalize, (place (on (oil jug) stool)) }

(pick on), internalize, internalize, (place (on on (oil jug))), { [(remember (pick (oil jug))), (remember externalize)], (then (remember (pick (oil jug))) (remember externalize)) }, internalize, (place (on jug (on tire stool))), internalize, internalize, internalize, (pick (on (oil jug) stool)), internalize, (pick (on (oil jug) workbench)), (place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) (want i (on (oil jug) stool)))), internalize, internalize, (forget (forget (remember (pick tire)))), internalize, (remember (pick stool)), { [(pick want), (remember (place (on stool ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))))))], (then (pick want) (remember (place (on stool ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))))))) }, (pick (want i (on (oil jug) stool))), (pick stool), (pick oil), (pick stool), internalize, internalize, (pick tire), { [(place (on oil tire)), (pick (on tire (left gripper)))], (then (place (on oil tire)) (pick (on tire (left gripper)))) }, (place (on workbench (on tire (left gripper)))), (place (on stool (on tire (left gripper)))), (forget (forget (pick (left gripper)))), internalize, (pick (oil jug)), (forget externalize), (remember internalize), (place (on workbench jug)), { [externalize, (place (on oil tire))], (then externalize (place (on oil tire))) }, { internalize, (place (on (oil jug) stool)) }

---

internalize, internalize, internalize, internalize, (pick (on (oil jug) workbench)), internalize, internalize, internalize, internalize, (remember externalize), (remember (then (remember externalize) (place (on on (on (oil jug) stool)))))), internalize, (remember externalize), internalize, (pick (on (oil jug) stool)), internalize, internalize, { [(place (on (on tire stool) jug)), { [internalize, (remember externalize)], (then internalize (remember externalize)) }], (then (place (on (on tire stool) jug)) { [internalize, (remember externalize)], (then internalize (remember externalize)) }) }, internalize, (forget internalize), { [(place (on (want i (on (oil jug) stool)) (on (oil jug) workbench))), externalize], (then (place (on (want i (on (oil jug) stool)) (on (oil jug) workbench))) externalize) }, (place (on (on (oil jug) stool) want)), { internalize, (place (on (oil jug) stool)) }, (place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) oil)), (forget (pick want)), { internalize, (pick (oil jug)) }, (pick gripper), { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }

{ [(pick jug), (pick (on tire stool))], (then (pick jug) (pick (on tire stool))) }, internalize, (place (on oil stool)), (pick (on (oil jug) stool)), (place (on jug (on (oil jug) workbench))), { [(place (on stool (oil jug))), (forget (forget (place (on stool jug))))], (then (place (on stool (oil jug))) (forget (forget (place (on stool jug))))) }, (place (on oil stool)), internalize, { [internalize, internalize], (then internalize internalize) }, { [(pick stool), (place (on (oil jug) want))], (then (pick stool) (place (on (oil jug) want))) }, (place (on oil stool)), (place (on i want)), (forget (then (forget (place (on (on (oil jug) workbench) (want i (on (oil jug) stool))))) (pick on))), { [(forget (remember (remember (remember externalize)))), internalize], (then (forget (remember (remember (remember externalize)))) internalize) }, { [internalize, (pick (want i (on (oil jug) stool)))], (then internalize (pick (want i (on (oil jug) stool))))) }, (pick (on tire stool)), (place (on stool on)), (place (on oil jug)), internalize, (place (on stool (oil jug))), (place (on oil jug)), (place (on oil stool)), (place (on oil oil)), (place (on oil stool)), (place (on (on tire stool) workbench)), { [(pick jug), (pick stool)], (then (pick jug) (pick stool)) }, (place (on oil stool)), { [(forget (pick want)), (place (on (want i (on (oil jug) stool)) stool))], (then (forget (pick want)) (place (on (want i (on (oil jug) stool)) stool))) }, internalize, (pick (want i (on (oil jug) stool))), internalize, (remember (pick (on (oil jug) stool))), internalize, (place (on oil on)), { [externalize, (pick stool)], (then externalize (pick stool)) }, { internalize, (place (on (oil jug) stool)) }, { [(remember (forget (forget externalize))), (remember (forget (forget externalize)))], (then (remember (forget (forget externalize))) (remember (forget (forget externalize)))) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }

---

internalize, internalize, internalize, { [(place (on want on)), (pick tire)], (then (place (on want on)) (pick tire)) }, (place (on (oil jug) left)), (place (on (on nothing stool) (on (oil jug) workbench))), (remember internalize), internalize, internalize, externalize, { internalize, (place (on (oil jug) stool)) }, externalize, { internalize, (place (on (oil jug) stool)) }, (place (on ((on (oil jug) workbench) (on nothing stool) (on tire (left gripper)) (want i (on (oil jug) stool)) (remember i internalize)) internalize)), (place (on stool jug)), { [(pick tire), (pick nothing)], (then (pick tire) (pick nothing)) }, { internalize, (place (on (oil jug) stool)) }, (place (on (remember i internalize) (remember i internalize))), { [(pick gripper), (pick right)], (then (pick gripper) (pick right)) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }

(forget (then (remember (remember (pick i))) (pick stool))), internalize, internalize, internalize, internalize, internalize, internalize, (place (on oil want)), (pick (oil jug)), (place (on tire (oil jug))), internalize, internalize, internalize, { [internalize, (remember (pick nothing))], (then internalize (remember (pick nothing))) }, externalize, (pick (oil jug)), { internalize, (place (on (oil jug) stool)) }

(remember (pick oil)), (pick (oil jug)), (place (on (want i (on (oil jug) stool)) ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool))))), internalize, internalize, (pick (oil jug)), internalize, (pick (oil jug)), (forget externalize), internalize, internalize, (pick (oil jug)), (place (on gripper (right gripper))), internalize, internalize, externalize, (place (on workbench (oil jug))), { [(pick ((on nothing workbench) (on tire stool) (on (oil jug) (right gripper)) (want i (on (oil jug) stool)))), (place (on right on))], (then (pick ((on nothing workbench) (on tire stool) (on (oil jug) (right gripper)) (want i (on (oil jug) stool)))) (place (on right on))) }, (pick (oil jug)), (pick (oil jug)), (pick (oil jug)), externalize, (place (on (left gripper) gripper)), (forget (pick (on tire stool))), (pick (oil jug)), externalize, (forget (forget (place (on jug ((on nothing workbench) (on tire stool) (on (oil jug) (right gripper)) (want i (on (oil jug) stool))))))), externalize, (pick (oil jug)), (place (on gripper workbench)), (pick (oil jug)), (place (on (on (oil jug) (right gripper)) jug)), (pick (oil jug)), { internalize, (place (on (oil jug) stool)) }

---

externalize, (pick (oil jug)), externalize, { internalize, (place (on (oil jug) stool)) }

(pick (oil jug)), (pick (oil jug)), internalize, (pick (oil jug)), externalize, { internalize, (pick (oil jug)) }, (pick (right gripper)), (pick (on tire stool)), (remember (then internalize externalize)), externalize, { internalize, (pick (oil jug)) }, (remember internalize), { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }

(pick (oil jug)), (pick (oil jug)), (remember (then (place (on gripper stool)) (pick jug))), (place (on workbench ((on nothing workbench) (on tire stool) (on (oil jug) (right gripper)) (want i (on (oil jug) stool))))), (pick (on (oil jug) (right gripper))), (forget (place (on jug right))), internalize, (pick (oil jug)), internalize, externalize, { internalize, (pick (oil jug)) }, (pick (want i (on (oil jug) stool))), (pick (oil jug)), (place (on (oil jug) (on (oil jug) (left gripper)))), externalize, externalize, (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), (remember (place (on (on (oil jug) stool) stool))), (pick (on (oil jug) stool)), (forget externalize), { internalize, (pick (oil jug)) }, externalize, (place (on (oil jug) oil)), (pick (on (oil jug) stool)), { [(remember internalize), (remember (pick (on (oil jug) (right gripper)))))], (then (remember internalize) (remember (pick (on (oil jug) (right gripper))))) }, (pick ((on nothing workbench) (on tire stool) (on (oil jug) (right gripper)) (want i (on (oil jug) stool)) (remember i internalize) (remember i (pick (on (oil jug) (right gripper)))))), { internalize, (place (on (oil jug) stool)) }

---

internalize, internalize, (pick (oil jug)), (pick (on (oil jug) stool)), (place (on (oil jug) jug)), internalize, (place (on (oil jug) stool))

---

internalize, (pick (on (oil jug) stool)), (forget (place (on tire oil))), { [(pick oil), (place (on (on tire stool) want))], (then (pick oil) (place (on (on tire stool) want))) }, (pick (oil jug)), (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), (place (on (left gripper) left)), (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), (forget (forget externalize)), (pick (on (oil jug) stool)), (remember externalize), (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), (pick ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)))), (forget externalize), (forget internalize), (pick ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)))), (pick (on (oil jug) stool)), (forget internalize), internalize, (pick (on (oil jug) stool)), (pick (on (oil jug) (left gripper))), (pick (on (oil jug) stool)), internalize, internalize, (place (on jug (on tire stool))), (place (on (oil jug) stool))

(pick (on (oil jug) stool)), (pick (on (oil jug) stool)), externalize, { internalize, (pick (oil jug)) }, externalize, (pick (on (oil jug) (left gripper))), (pick (on (oil jug) stool)), (place (on oil (on (oil jug) stool))), (pick (on (oil jug) stool)), (pick (on (oil jug) stool)), { internalize, (pick (oil jug)) }, (forget (remember externalize)), { internalize, (place (on (oil jug) stool)) }

(place (on (on (oil jug) stool) tire)), (remember (remember externalize)), externalize, externalize, (pick (on (oil jug) stool)), { internalize, (place (on (oil jug) stool)) }, (place (on oil jug)), externalize, { internalize, (place (on (oil jug) stool)) }, (place (on (oil jug) jug)), (place (on (want i (on (oil jug) stool)) jug)), { internalize, (place (on (oil jug) stool)) }, (remember (pick want)), (forget (remember (forget (place (on (on (oil jug) stool) (on (oil jug) stool)))))), (pick (on (oil jug) stool)), { [(pick (on (oil jug) stool)), { [(place (on tire want)), { internalize, (pick (oil jug)) }], (then (place (on tire want)) { internalize, (pick (oil jug)) }) }], (then (pick (on (oil jug) stool)) { [(place (on tire want)), { internalize, (pick (oil jug)) }], (then (place (on tire want)) { internalize, (pick (oil jug)) }) }) }), (forget (forget (place (on tire jug)))), { internalize, (pick (oil jug)) }, externalize, (forget externalize), externalize, (remember (place (on tire jug))), (forget (pick (oil jug))), (place (on (oil jug) stool))

---

internalize, (place (on stool (oil jug))), (place (on (oil jug) stool)), { [externalize, { [{ internalize, (place (on (oil jug) stool)) }, (pick i)], (then { internalize, (place (on (oil jug) stool)) } (pick i)) }], (then externalize { [{ internalize, (place (on (oil jug) stool)) }, (pick i)], (then { internalize, (place (on (oil jug) stool)) } (pick i)) }) }, externalize, { internalize, (pick (oil jug)) }, (place (on (oil jug) jug)), (forget (pick nothing)), (remember (then externalize (pick (on nothing workbench)))), { internalize, (place (on (oil jug) stool)) }

---

internalize, (place (on stool (oil jug))), (place (on (oil jug) stool)), externalize, externalize, { internalize, (pick (oil jug)) }, (place (on (on (oil jug) stool) jug)), externalize, (place (on gripper stool)), (place (on (want i (on (oil jug) stool)) (on tire stool))), (remember (pick (on (oil jug) stool))), (forget (forget (pick (on nothing workbench)))), (forget (remember externalize)), { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }

(forget (then (remember internalize) (pick (on tire stool)))), (forget (place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) (on tire stool)))), internalize, externalize, externalize, (place (on (oil jug) stool)), (place (on (oil jug) stool)), { [{ internalize, (place (on (oil jug) stool)) }, { [(pick (on (oil jug) stool)), (place (on want (on (oil jug) stool)))], (then (pick (on (oil jug) stool)) (place (on want (on (oil jug) stool)))) }], (then { internalize, (place (on (oil jug) stool)) } { [(pick (on (oil jug) stool)), (place (on want (on (oil jug) stool)))], (then (pick (on (oil jug) stool)) (place (on want (on (oil jug) stool)))) }) }, (place (on (oil jug) stool)), (place (on (on (oil jug) stool) stool)), (place (on (on (oil jug) stool) stool)), externalize, externalize, externalize, (place (on ((on (oil jug) workbench) (on tire stool) (want i (on (oil jug) stool))) stool)), externalize, { internalize, (place (on (oil jug) stool)) }, (remember (forget (remember internalize))), { internalize, (place (on (oil jug) stool)) }, (pick (oil jug)), { [{ internalize, (pick (oil jug)) }, (remember (pick (left gripper)))], (then { internalize, (pick (oil jug)) } (remember (pick (left gripper)))) }, (place (on (oil jug) jug)), { internalize, (place (on (oil jug) stool)) }

(pick (oil jug)), (place (on (oil jug) left)), (place (on jug (oil jug))), internalize, internalize, internalize, (place (on (oil jug) stool))

(place (on jug oil)), (place (on (on (oil jug) workbench) (on tire stool))), externalize, (pick (oil jug)), (place (on (oil jug) jug)), (forget (place (on jug (on tire stool)))), (remember externalize), (place (on want (remember i externalize))), (remember (pick (on (oil jug) stool))), (remember (pick tire)), (remember (remember (place (on (on (oil jug) stool) ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)) (remember i externalize)))))), { [(pick (on nothing workbench)), (place (on ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)) (remember i externalize)) jug))], (then (pick (on nothing workbench)) (place (on ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)) (remember i externalize)) jug))) }, (place (on (on (oil jug) stool) gripper)), (place (on (on (oil jug) stool) stool)), (remember (then (forget internalize) internalize)), (place (on (oil jug) stool))

(pick (oil jug)), (place (on (oil jug) stool))

(pick (oil jug)), (place (on (oil jug) stool))

(pick (oil jug)), (forget externalize), internalize, (forget externalize), internalize, (forget (pick jug)), (forget (then (forget externalize) internalize)), internalize, externalize, { [(forget externalize), (place (on left (oil jug)))], (then (forget externalize) (place (on left (oil jug)))) }, externalize, (forget (pick left)), (place (on gripper jug)), (forget (remember externalize)), (place (on (oil jug) gripper)), { internalize, (place (on (oil jug) stool)) }

---

externalize, { internalize, (place (on (oil jug) stool)) }, (pick (oil jug)), (place (on (on (oil jug) stool) on)), { internalize, (place (on (oil jug) stool)) }

---

internalize, { [(remember internalize), (place (on on tire))], (then (remember internalize) (place (on on tire))) }, (forget (place (on remember tire))), (pick (oil jug)), { [(remember (place (on (on nothing workbench) nothing))), (pick jug)], (then (remember (place (on (on nothing workbench) nothing))) (pick jug)) }, { [(place (on tire stool)), (pick jug)], (then (place (on tire stool)) (pick jug)) }, internalize, (remember (place (on gripper left))), internalize, (place (on (oil jug) (on nothing workbench))), (pick nothing), externalize, (place (on (on (oil jug) (left gripper)) (want i (on (oil jug) stool)))), { [{ [externalize, (forget (place (on internalize tire)))], (then externalize (forget (place (on internalize tire))))) }, (pick stool)], (then { [externalize, (forget (place (on internalize tire)))], (then externalize (forget (place (on internalize tire)))) } (pick stool)) }, (forget (pick jug)), { [(remember externalize), (pick stool)], (then (remember externalize) (pick stool)) }, (place (on (oil jug) stool))

{ [(pick workbench), internalize], (then (pick workbench) internalize) }, (pick (on (oil jug) stool)), internalize, (pick (oil jug)), externalize, externalize, (forget (pick (oil jug))), (place (on workbench gripper)), { [(place (on on (on (oil jug) stool))), { internalize, (place (on (oil jug) stool)) }], (then (place (on on (on (oil jug) stool))) { internalize, (place (on (oil jug) stool)) }) }

(pick (oil jug)), externalize, (forget (then (remember (place (on ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool))) oil))) internalize)), externalize, (forget (pick stool)), (place (on gripper gripper)), { [(remember externalize), (pick stool)], (then (remember externalize) (pick stool)) }, (place (on (oil jug) stool))

(pick (oil jug)), (place (on (oil jug) stool))

---

externalize, (place (on tire jug)), { internalize, (pick (oil jug)) }, externalize, { [(place (on oil on)), (pick (left gripper))], (then (place (on oil on)) (pick (left gripper))) }, (pick (oil jug)), { internalize, (pick (oil jug)) }, (pick oil), { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
(pick (oil jug)), externalize, (place (on gripper stool)), { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, (forget (pick (left gripper))), externalize, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (pick (oil jug)) }, { internalize, (place (on (oil jug) stool)) }
internalize, (pick (oil jug)), { [(place (on stool nothing)), (pick i)], (then (place (on stool nothing)) (pick i)) }, (place (on (oil jug) stool))
externalize, (remember externalize), (place (on (remember i externalize) (want i (on (oil jug) stool)))), (place (on (oil jug) stool)), (pick (oil jug)), (forget (pick (on tire stool))), { [(remember externalize), (remember (place (on ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)) (remember i externalize)) (on (oil jug) stool))))], (then (remember externalize) (remember (place (on ((on nothing workbench) (on tire stool) (on (oil jug) (left gripper)) (want i (on (oil jug) stool)) (remember i externalize)) (on (oil jug) stool))))) }, { internalize, (pick (oil jug)) }, (place (on i (want i (on (oil jug) stool)))), { internalize, (pick (oil jug)) }, externalize, (remember internalize), (place (on (oil jug) stool))
(place (on (on (oil jug) workbench) stool)), internalize, externalize, (pick (oil jug)), { [(place (on jug gripper)), externalize], (then (place (on jug gripper)) externalize) }, { internalize, (place (on (oil jug) stool)) }
internalize, (pick (oil jug)), (place (on (oil jug) stool))

References

Altshuler, D., Parsons, T., & Schwarzschild, R. (2019). *A course in semantics (draft version)*. Cambridge, MA: MIT Press.

Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent neural network grammars. *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 199-209.

Fedorenko, E., & Varley, R. (2016). Language and thought are not the same thing: Evidence from neuroimaging and neurological patients. *Annals of the NY Academy of Sciences*, *1369*, 132-53.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd international joint conference on artificial intelligence* (p. 608-620). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Fodor, J. A. (1975). *The language of thought*. Cambridge, MA: Harvard University Press.

Gleitman, L. (1990). The structural sources of verb meanings. *Language Acquisition*, *1*(1), 3-55.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems 27* (pp. 2672–2680).

Hausknecht, M. J., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *CoRR*, *abs/1507.06527*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735-1780.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, *35*, 73-101.

James, G. S. (1954). Tests of linear hypotheses in univariate and multivariate analysis when the ratios of the population variances are unknown. *Biometrika*, *41*(1-2), 19-43.

Katz, B. (1997). Annotating the world wide web using natural language. In *Proceedings of the 1997 computer-assisted information searching on the internet conference* (p. 136-155). Montreal, Canada: Centre de Hautes Etudes Internationales d'Information Documentaire.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* (Poster session presented at the Third International Conference for Learning Representations, San Diego, CA)

Minsky, M. (1986). *The society of mind.* New York, NY, USA: Simon & Schuster, Inc.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529-533.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (p. 1532-1543). Doha, Qatar: The Association for Computational Linguistics.

Pinker, S. (1984). *Language learnability and language development.* Cambridge, MA: Harvard University Press.

Ranzato, M., Huang, F., Boureau, Y., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 ieee computer society conference on computer vision and pattern recognition, cvpr'07.*

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, *abs/1503.00075*.

Thrush, T., & Winston, P. (2018). The partial mental state inducer: learning intuition with few training examples and k-line theory. *Advances in Cognitive Systems*, *7*, 97-116.

Winston, P. (1970). *Learning structural descriptions from examples* (Tech. Rep.). Cambridge, MA, USA.

Winston, P. (2012). The next 50 years: a personal view. *Biologically Inspired Cognitive Architectures*, *1*, 92–99.

Winston, P. (2018). *Self-aware problem solving* (Tech. Rep.). Cambridge, MA, USA.

Winston, P., & Holmes, D. (2018). *The genesis enterprise: Taking artificial intelligence to another level via a computational account of human story understanding* (Tech. Rep.). Cambridge, MA, USA.