Title
Utility Learning, Non-Markovian Planning, and Task-Oriented Programming Language

Author
Shukla, Nishant

Publication Date
2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Utility Learning, Non-Markovian Planning, and Task-Oriented Programming Language

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Nishant Shukla

2019

ABSTRACT OF THE DISSERTATION

Utility Learning, Non-Markovian Planning,

and Task-Oriented Programming Language

by

Nishant Shukla

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2019

Professor Song-Chun Zhu, Chair

We formulate a domain-independent language for representing stochastic tasks, and show how this representation can be synthesized from few training observations. In one experiment, we study how a physical robot may learn to fold clothes from a small number of visual observations, in contrast to big data techniques. Under the same framework, we also show how a virtual chat-bot may learn dialogue policies from few example transcripts, resulting in an interpretable dialogue model, outperforming current statistical techniques. Central to both examples in the concept of utility, why it's essential for generalizability, and how to learn it from small data.

The dissertation of Nishant Shukla is approved.

Demetri Terzopoulos

Guy Van den Broeck

Yingnian Wu

Song-Chun Zhu, Committee Chair

University of California, Los Angeles

2019

To my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Professor Song-Chun Zhu for challenging and trusting me throughout the years. I'm also grateful for all the inspiration from my mentors: Caiming Xiong and Yixin Zhu. Lastly, thank you everyone from the Center for Vision, Cognition, Learning, and Autonomy (VCLA) for your engaging discussions.

Chapter One is a version of *A unified framework for human-robot knowledge transfer* from 2015 AAAI Fall Symposium Series, by Shukla, N., Xiong, C., & Zhu, S. C.

Chapter Two is a version of *Robot learning with a spatial, temporal, and causal and-or graph* from 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2144-2151) by Xiong*, C., Shukla*, N., Xiong, W., & Zhu, S. C.

Chapter Three is a version of *Task learning through visual demonstration and situated dialogue* from Workshops at the Thirtieth AAAI Conference on Artificial Intelligence by Liu, C., Chai, J. Y., Shukla, N., & Zhu, S. C. (2016, March).

Chapter Four is a version of *Learning human utility from video demonstrations for deductive planning in robotics* from Conference on Robot Learning (CoRL) (pp. 448-457) by Shukla, N., He, Y., Chen, F., & Zhu, S. C. (2017, October).

Chapter Five is a version of *Utility learning, non-Markovian planning, and task-oriented programming language* in submission to Artificial Intelligence by Shukla, N., Wang, P., Gao, F., Zhang, V. & Zhu, S. C. (2019, May).

Chapter Six is a version of *Dialogue manager programming language* from W3C Community Group Final Report (`https://www.w3.org/2019/04/dmpl`) by Shukla, N., Solano, N., & Zhang, V. (2019, April).

Portions of this work were supported by the Office of Naval Research grant N000141010933, the DARPA Award N66001-15- C-4035, the DARPA-XAI Project Award No. N66001-17-2-4029, the DARPA SIMPLEX program N66001-15-C-4035, the DARPA MSEE project FA 8650-11-1-7149, and the National Science Foundation IIS-1208390 and IIS-1617682.

VITA

2016	M.S. Computer Science, UCLA
2016 Summer	Research Engineer Intern, SpaceX
2015 Summer	Research Engineer, UCLA
2014 Summer	Software Developer Intern, Foursquare
2010-2014	B.S. Computer Science, B.A. Mathematics, University of Virginia
2013 Summer	Software Developer Intern, Facebook
2012 Summer	Software Developer Intern, Microsoft
2011 Summer	Software Developer Intern, WillowTree
2010 Summer	Software Developer Intern, Buchanan & Edwards

PUBLICATIONS

A unified framework for human-robot knowledge transfer. 2015 AAAI Fall Symposium Series. **Shukla, N.**, Xiong, C., & Zhu, S. C. (2015, September).

Joint learning from video and caption. ICCV15: Closing the Loop Between Vision and Language. Wang, T., **Shukla, N.**, Xiong C. (2015, December).

Robot learning with a spatial, temporal, and causal and-or graph. 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2144-2151). Xiong*, C., **Shukla*, N.**, Xiong, W., & Zhu, S. C. (2016, May).

A Unified Knowledge Representation System for Robot Learning and Dialogue. University of California, Los Angeles. **Shukla, N.** (2016).

Task learning through visual demonstration and situated dialogue. Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. Liu, C., Chai, J. Y., **Shukla, N.**, & Zhu, S. C. (2016, March).

Jointly learning grounded task structures from language instruction and visual demonstration. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1482-1492). Liu, C., Yang, S., Saba-Sadiya, S., **Shukla, N.**, He, Y., Zhu, S. C., & Chai, J. (2016).

Learning human utility from video demonstrations for deductive planning in robotics. Conference on Robot Learning (CoRL) (pp. 448-457). **Shukla, N.**, He, Y., Chen, F., & Zhu, S. C. (2017, October).

Machine learning with TensorFlow. Manning Publications Co. **Shukla, N.** (2018).

Dialogue manager programming language. W3C Community Group Final Report (`ttps://www.w3.org/2019/04/dmpl`) by **Shukla, N.**, Solano, N., & Zhang, V. (2019, April).

Utility learning, non-Markovian planning, and task-oriented programming language. In submission: Artificial Intelligence. **Shukla, N.**, Wang, P., Gao, F., Zhang, V. & Zhu, S. C. (2019, May).

CHAPTER 1

Introduction

The fundamental problem we're addressing is how to represent a task for execution on a robot. However, that problem description, stated exactly as is, does not specify a success criteria for representation nor execution. Plenty of details need to be disambiguated: What is a task? What does it mean for a task be to executed? How can we know whether a robot learned the task?

Therefore, in this first chapter, we define a litmus test called the Knowledge Transfer Test (KTT) that also serves as an organizational tool to visualize all our contributions into a cohesive picture. Subsequent chapters are assembled to fit nicely into the mental map set up by the KTT. Specifically, Figure 1.4 captures the complete knowledge transfer pipeline that the rest of the chapters aim to solve using utility learning, non-Markovian planning, and task-oriented programming language representation.

We begin by proposing a graphical structure, called an And-Or graph, to represent knowledge or skills. The And-Or graph representation may model spatial, temporal, or causal relationships. We hand-design a cloth-folding task using this representation for one robot and then test it running on a different type of robot. This proof by example sets the scene for deeper investigations, covered by Chapter 2.

1.1 Human-Robot Knowledge Transfer

Transferring knowledge is a vital skill between humans for efficiently learning a new concept. In a perfect system, a human demonstrator can teach a robot a new task by using natural language and physical gestures. The robot would gradually accumulate and refine its spatial, temporal, and causal understanding of the world. The knowledge can then be transferred back to another hu-

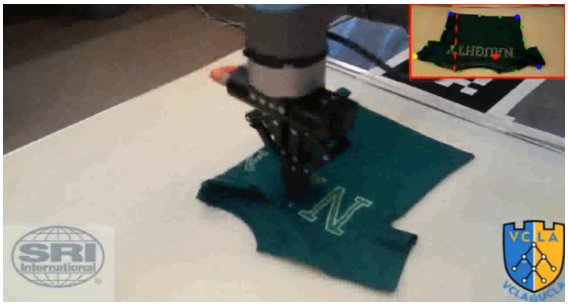


Figure 1.1: The robot performs a cloth folding task after learning from a human demonstration.

man, or further to another robot. The implications of effective human to robot knowledge transfer include the compelling opportunity of a robot acting as the teacher, guiding humans in new tasks.

The technical difficulty in achieving a robot implementation of this caliber involves both an expressive knowledge structure and a real-time system for non-stop learning and inference. Recently, skill acquisition and representation have become some of the core challenges in achieving robots capable of learning through human demonstration.

We propose a real-time unified learning and inference framework for knowledge acquisition, representation, and transfer. Knowledge is represented in a Spatial, Temporal, and Causal And-Or Graph (STC-AOG) hierarchical network [1], which can be thought of as a stochastic grammar. The STC-AOG encapsulates the hierarchical compositional structures of physical objects, logical deductions, and instance-based actions. Our knowledge graph manipulation framework enables learning to be a continuous on-line process that occurs alongside inference. We view a robot as a knowledge database, where humans may deposit and withdraw skills. These skills can be used by both humans and robots alike.

As a proof of concept, we teach an industrial robot how to fold clothes (Figure 1.1). The robot watches a human demonstrator and learns in real-time. To test the faithfulness of the human-robot knowledge transfer, we propose an evaluation procedure called the Knowledge Transfer Test. Our experiments demonstrate that our proposed framework can adequately transfer knowledge to and from a robot. Furthermore, we illustrate our system's interactive learning capabilities that are backed by a Bayesian formulation.

1.1.1 Related Works

We extend the learning of And-Or grammars and semantics from video [2] to an interactive real-time robotics platform with a natural communication interface between humans. The And-Or data structure has also been used in learning a visually grounded storyline model from labeled videos [3]; however, our system requires no labeled data, and evokes a richer segmentation of spatial, temporal, and causal concepts for more tractable queries. Miller, Van Den Berg, Fritz, Darrell, Goldberg, and Abbeel [4] establish high standards for a cloth-folding robot, but our focus is instead on novel learning, knowledge representation, and knowledge transfer. The action-planning inference system in our STC-AOG data structure resembles closest to a Planning Graph [5], which is essentially an STC-AOG without causal nodes. Yang, Li, Fermuller, and Aloimonos [6] learn concrete action commands from small video clips. Unlike their system, our design allows a modifiable grammar and our performance is measured on multi-step actions.

1.2 Representation

We encapsulate knowledge by an expressive graphical data structure $G_\Omega = (G_s, G_t, G_c)$ which models the compositional structures of objects G_s, actions G_t and causality G_c. A specific piece of information or skill, such as how to fold clothes, is a subgraph $G \subseteq G_\Omega$. The goal of knowledge transfer is to deliver G from one agent (e.g. human) to another (e.g. robot) with a minimum loss of knowledge.

In human-robot interactions, we restrict communication to only physical actions through a video-camera sensor V, and natural language text L. Therefore, the learner must construct an optimal G based only on V and L, resulting in the Bayesian formulation,

$$G^* = \arg\max_{G_t} P(G_t|V, L) = \arg\max_{G_t} \frac{P(V|G_t, L)P(G_t, L)}{P(V, L)}$$

Similar to Ha, Kim, and Zhang [7], we use a graph Monte Carlo method that assumes the graph

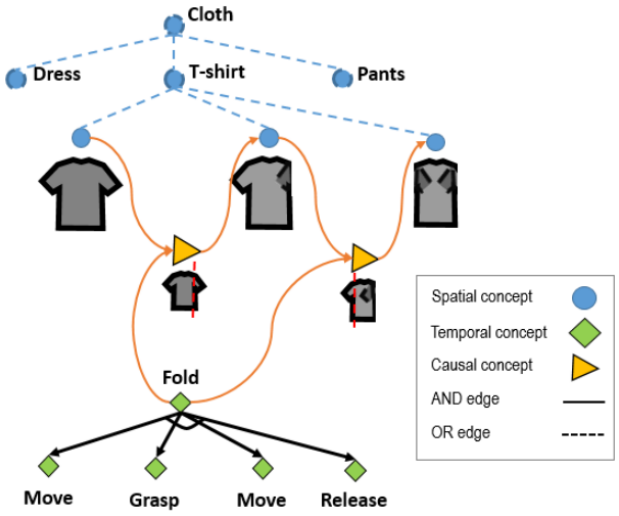


Figure 1.2: STC-AOG for a cloth-folding task.

structure is determined only by that of the previous iteration.

$$G^* = \arg\max_{G_t} P(V|G_t, L)P(G_{t-1}, L)$$

The learning algorithm is similar to a marginalization-based parameter learning algorithm, where we first marginalize our STC-AOG, and learn the S-AOG, T-AOG and C-AOG separately, then jointly learn the conditional model between each other.

Figure 1.2 shows a small segment of G^*, and specific details of the spatial, temporal, and causal segments are described as follows.

1.2.1 Spatial Representation

Sensory data from the environment is encoded to form a belief representation. We use a PrimeSense camera to capture RGB-D (Red, Green, Blue, and Depth) information per frame. We represent every cloth by a high-level abstract understanding based off its contour shape, and a low-level representation by specific keypoints. The keypoints and contour shape data are used as input to the folding algorithm which generalizes to arbitrary articles of clothing. To store the hierarchical structure of physical objects, we use an And-Or Graph data-structure, called the Spatial And-Or

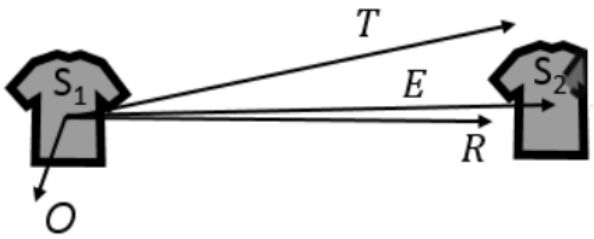


Figure 1.3: Forces associated with the simulated actuator T and other forces O sum to produce a resulting R force, which is nearly collinear to the observed end-state E, implying that T causes S_1 to become S_2.

Graph (SAOG) [8]. AND nodes in the S-AOG represent structural compositionality (i.e. a vehicle has an engine). OR nodes in the S-AOG represent variations (i.e. a car is a type of vehicle).

1.2.2 Causal Representation

The perceived model of the world is then used to learn a logical cause-and-effect type of reasoning from a singleinstance, inspired by the Dynamics Model [9].

The Dynamics Model defines causal relationships as interpretations of force vectors. The nodes in the S-AOG are normalized feature vectors in a higher dimensional space, and are acted on by force vectors from the T-AOG. As per the model, if the net force on a spatial node is collinear with the vector represented by the end-state of an action, then a causality is deduced, as shown in Figure 1.3.

The causal relationships are stored in a Causal And-Or Graph (C-AOG). AND nodes in the C-AOG indicate that all preconditions are necessary, whereas OR nodes indicate that only one of the preconditions is sufficient.

1.2.3 Temporal Representation

These deductive models are used to plan out the next course of action, which may affect the environment. The actuators that affect the environment, whether by the robot or the human, are represented in another data-structure, called the Temporal And-Or Graph (T-AOG). AND nodes

represent actions done in a series of steps. OR nodes represent variations in possible actions.

1.2.4 Joint Representation

We represent the physical models (S-AOG), the reasoning models (C-AOG), and the environment actuators (T-AOG) all into one unified Spatial Temporal Causal And-Or Graph (STC-AOG) data structure. As a consequence, the whole system forms a closed-loop from perception to learning to inference, and back again to perception. Figure 2 demonstrates a small portion of the STC-AOG applied to a clothfolding task.

1.3 Knowledge Transfer Test

One of the most useful properties of knowledge transfer is the ability to propagate the knowledge among others. To determine the proficiency of knowledge transfer to and from an artificial agent, we propose the following three-part test.

A human demonstrator H_A will perform a chosen task to receive a task score s_0 by a human judge. In the first part of the test, H_A will teach this task to a robot R_A that has not been previously trained on the task. The judge will assign a task score s_1 based on R_A's performance.

Next, the second test will evaluate R_A's ability to transfer knowledge to another robot R_B that has not been previously trained on the task. Robot-to-robot knowledge transfer can be as direct as sending over the explicit knowledge structure, which in our case is the STC-AOG. Again, the judge will assign a task score s_2.

Finally, R_B must teach the task to a human H_B that has no previous knowledge of the task procedure. A task score s_3 will be assigned by the judge. If all three task scores match within 10% of s_0, then R_A is said to have passed the Knowledge Transfer Test (KTT). The entire process is visualized in Figure 1.4.

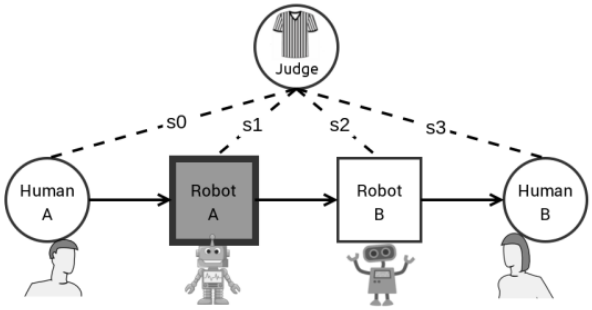


Figure 1.4: Arrows represent the direction of knowledge transfer. The judge assigns task scores at each step.

1.4 Experimental Results

We evaluate our framework on a two-armed robot using the proposed Knowledge Transfer Test on a cloth folding task. To benchmark real-time performance, we calculate the ratio between the duration of the demonstration and the total time spent learning. The average speed of our robot system is 5 fps, resulting in a system which out-performs most perception-heavy robot learning-systems today.

Our robot was able to understand the cloth-folding task, generating a STC-AOG similar to Figure 2, confidently enough to pass the first part of the KTT. We were able to save the graphical structure and load it into a different type of robot to pass the second part of the KTT. The robot was also able to teach the task successfully to a human, but since folding clothes is already a well known skill by most humans, we set aside deeper investigation of robot-to-human teaching for future work.

CHAPTER 2

Learning from Demonstration

Knowledge may be acquired through learning from demonstrations (LfD), which will lay the foundations for utility learning and task-oriented programming languages. LfD is one way to accomplish the first step of The Knowledge Transfer Test described in the last chapter. Concretely, in our experiments, a human demonstrates a cloth-folding task, by performing a sequence of actions in front of a video recording device. The system models the spatial, temporal, and causal relationships.

With the Chomsky hierarchy of formal grammars, we can study the Spatial, Temporal, and Causal And-Or graph (STC-AOG) as a (stochastic) context-free grammar. Doing so reminds us that the STC-AOG has the expressive power to represent tasks in ways a finite-state machine (or regular grammar) may never fully capture. In a later chapter, we'll use this grammar perspective to upgrade the STC-AOG into a programming language.

2.1 Spatial, Temporal, and Causal And-Or Graph

Writing automated software on robots is not nearly as robust as that on traditional computers. This is due to the heavy burden of matching software assumptions to physical reality. The complexities and surprises of the real world require robots to adapt to new environments and learn new skills to remain useful.

In robot automation, implicit motor control is widely used for learning from human demonstrations [10] [11] [12]. However, implicit motor control is insufficient for generalizing robot execution. For instance, a robot can imitate a human's demonstration to open a door; yet, it cannot execute a similar motion trajectory such as opening a window without the explicit representation

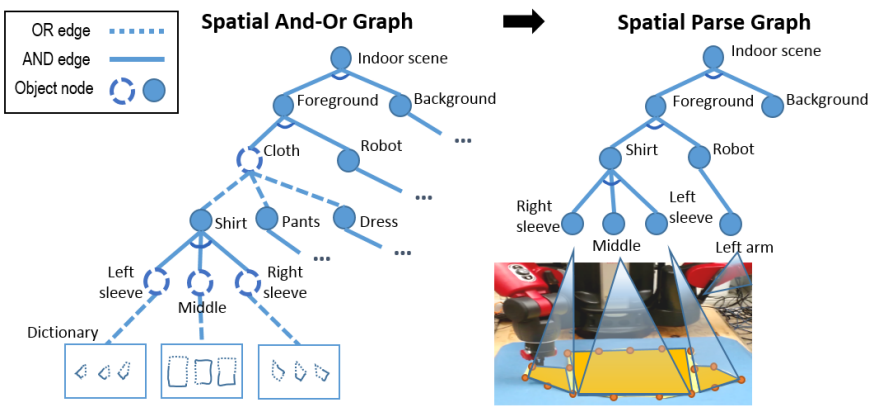


Figure 2.1: The Spatial And-Or Graph on the left represents the ongoing perceptual knowledge of the world, i.e. a learned stochastic visual grammar. A specific instance of the And-Or graph is realized in the parse graph on the right.

of the task. Intuition such as how to rotate the joints of an arm is not something easily expressible, but rather learned through experiences. Uniting explicit and implicit knowledge allows immediate communication through natural language [13], as well as clear grounding of abstract concepts into atomic actions.

In this chapter, we propose a unified framework to bridge the implicit motor control with explicit high-level knowledge so the robot can understand human behavior, perform a task with feedback control, and reason in vastly different environments. As a proof of concept, we teach a robot how to fold a shirt through few human demonstrations, and have it infer how to fold never-before-seen articles of clothing, such as pants or towels. The same causality-learning framework can be extrapolated to arbitrary tasks, not just cloth-folding. Specifically, the robot can learn different skills (e.g. flattening, stretching) depending on which features it tracks (e.g. smoothness, elastic stress). Moreover, since explicit knowledge is structured graphically, our framework naturally allows for the merging, trimming, and addition of knowledge from various human demonstrations, all with feedback control. The high-level concepts are human-understandable, so both the human and robot can communicate through this intermediate language [14]. Thus, programming the robot becomes an act of merely modifying a graph-based data structure.

2.2 Related Works

While precisely grounding a human demonstration to atomic robot actions has been done in various forms [6] [7] [15], we instead focus on the novel representation and generalizability of tasks. Beetz et al. integrate robot knowledge representation into the perception processes as well, but our framework allows alternative planning generated by probabilistic sampling to match observed expectations. For example, there are multiple ways to fold a t-shirt, and each of these ways has its own likelihood. Our probabilistic learning framework resembles closest to the human-inspired Bayesian model of imitation by Rao et la. [16]. However, we instead emphasize the hierarchical and ever-changing nature of spatial, temporal, and causal concepts in the real world.

Autonomously folding clothes has been demonstrated in various works. Wang et al. [17] were able to successfully design a perception-based system to manipulate socks for laundry. Miller et al. [4] have demonstrated sophisticated cloth-folding robots, and Doumanoglou et al. [18] have made substantial progress in autonomously unfolding clothes. On the other hand, our focus is to understand how to perform arbitrary tasks. There are other systems [6] that also learn concrete action commands from small video clips, but unlike those, our design allows a modifiable grammar and our performance is measured on multi-step long-term actions. Furthermore, our solution to knowledge representation is more powerful than commonsense reasoning employed by non-stochastic first-order logic [19], since it takes advantage of the probabilistic models under ambiguous real-world perception.

Our work is based on the knowledge representation system incorporated by Tu et al. [1], augmented heavily into the robotics domain. We extend the learning of event And-Or grammars and semantics from video [2] to our real-time robotics framework. The And-Or graph encapsulates a conformant plan under partial observability, enabling an architecture that is cognitively penetrable since an updated belief of the world alters the robot's behavior [20]. Unlike traditional graph planning [5], the hierarchical nature of the knowledge representation system enables a practical way of generating actions for a long-term goal.

2.3 Method

There is often a fine distinction between memorization and understanding, where the latter enables generalizing learned concepts. In order to understand a human task from demonstrations/videos such as cloth-folding, a knowledge representation system is necessary to ensure actions are not simply memorized. Four types of knowledge are important for understanding and generalizing:

- **Spatial knowledge** expresses the physical configuration of the environment when performing the task. For a cloth-folding task, a table, cloth, and each part of the cloth, such as the left and right sleeve of a shirt, needs to be detected.
- **Temporal knowledge** reveals the series of human actions in the process of the task. In cloth-folding, the hand motion, grip opening, and grip closing actions are essential. These actions combine together to form a fold action.
- **Causal knowledge** conveys the status change of an object in each dynamic human action. For example, a shirt may be folded in various ways, either by folding the left sleeve into the middle and then the right sleeve, or vice versa. Folding a cloth requires multiple hierarchical steps for reasoning.
- **The interplay between the spatial, temporal, and causal concepts** manifests a generalizable form of knowledge to be used in changing application domains. The robot must choose an action to achieve a state change by using a causal reasoning concept. Each of the three must work together to express learned knowledge.

2.3.1 Mathematical Formulation for Human Task

Given a set of human task demonstrations $\mathcal{D} = \{D_1, D_2, \cdots, D_n\}$ such as cloth-folding videos (i.e. series of RGBD images), the goal is to learn a joint model (G_{STC}) including Spatial, Tempo-

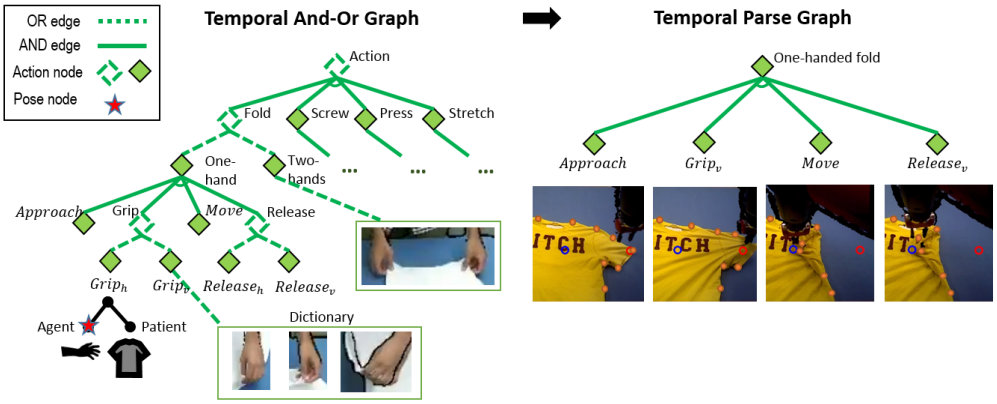


Figure 2.2: The Temporal And-Or Graph on the left is a database of all actions currently known in the real world. Each action has an associated agent and patient. The realized parse graph on the right shows a generated sequence of actions directly executable by the robot.

ral, and Causal concepts, that we formulate as

$$
\begin{aligned}
G^*_{STC} &= \underset{G_{STC}}{\operatorname{argmax}}\ P(G_{STC}|\mathcal{D}) \\
&= P(G_S|\mathcal{D}) \cdot P(G_T|\mathcal{D}) \cdot P(G_C|\mathcal{D}) \\
&\cdot P(R(G_S, G_T, G_C)|\mathcal{D})
\end{aligned}
\tag{2.1}
$$

where G_S is the model of spatial concepts, G_T is the model of temporal concepts, G_C is the model of causal concepts, and $R(G_S, G_T, G_C)$ is the relational/conditional model between spatial, temporal, causal concepts.

To implement this formulation, we need to define the concrete representation for each symbol in Eq. 1. Due to the structured and compositional nature of spatial, temporal, and causal concepts, we adopt the hierarchical stochastic grammar model, And-Or graph (AOG) [8], as the base of our model representation which is introduced below. To simplify the learning process, we marginalized the complex STC-AOG (G_{STC}) into the S-AOG (G_S), T-AOG (G_T) and C-AOG (G_C); thus, we can learn the G_S, G_T and G_C separately as the model's initialization, then jointly learn the conditional model between them.

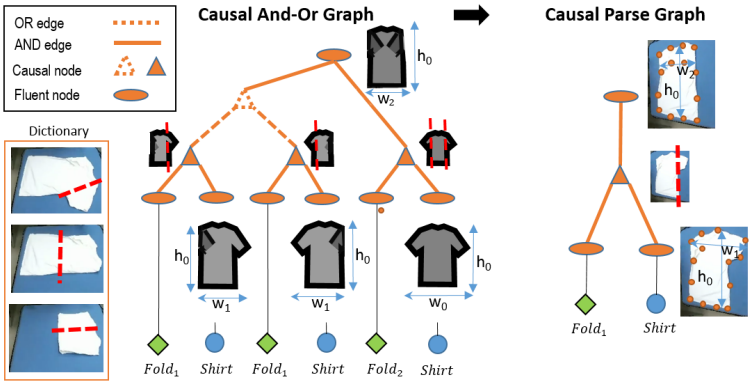


Figure 2.3: The Causal And-Or Graph encapsulates the fluent changes per action. The parse graph on the right shows the reasoning system in action.

2.3.2 And-Or Graph Overview

The And-Or Graph is defined as a 3-tuple $\mathcal{G} = (V, R, P)$, where $V = V^{AND} \cup V^{OR} \cup V^{T}$ consists of a disjoint set of And-nodes, Or-nodes, and Terminal nodes respectively. R is a set of relations between Or-nodes or subgraphs, each of which represents a generating process from a parent node to its children nodes. $P(r)$ is an expansion probability for each relation.

Figure 2.1 shows an example of an And-Or graph. An And-node represents the decomposition of a graph into multiple sub-graphs. It is denoted by an opaque circle, and all the out-going edges are opaque lines. An Or-node is a probabilistic switch deciding which of the sub-graphs to accept. It is denoted by an open circle with out-going edges drawn in dashed lines. The Terminal node represents grounded components, often referred to as a dictionary.

The nodes are structured into a hierarchical directed acyclic graph (DAG) structure. The AOG is a combination of a Markov tree and Markov random field, where an And-node corresponds to a graphic template model, and an Or-node corresponds to a switch in a Markov tree [21].

Given a set of human demonstrations $\mathcal{D}$, the graph $\mathcal{G}$ is composed of an AOG graph structure $\hat{\mathcal{G}}$ and parameters θ. The nodes and rules/edges in the graph structure aim to maximize the objective

function, denoted by the posterior probability:

$$P(\mathcal{G}|\mathcal{D}) = P(\hat{\mathcal{G}}, \theta|\mathcal{D}) \tag{2.2}$$

$$= P(\hat{\mathcal{G}}|\mathcal{D})P(\theta|\mathcal{D}, \hat{\mathcal{G}}) \tag{2.3}$$

The first term models the structure of an And-Or graph $\mathcal{G}$ from a human demonstration $\mathcal{D}$. To solve the first term, we manually design the structure of the S-AOG, but we learn the T-AOG and C-AOG structure automatically [2] [22] [23].

The second term models the parameters θ in the graph, given the learned knowledge graph structure. It is reformulated as follows:

$$P(\theta|\mathcal{D}, \hat{\mathcal{G}}) \propto \prod_{D_i \in \mathcal{D}} P(D_i|\theta, \hat{\mathcal{G}}) \tag{2.4}$$

$$\approx \prod_{D_i \in \mathcal{D}} \max_{pg_i} P(D_i|pg_i, \theta, \hat{\mathcal{G}})P(pg_i|\theta, \hat{\mathcal{G}}) \tag{2.5}$$

where pg_i is the parse graph of D_i. A parse graph is an instance of $\mathcal{G}$ where each Or-node decides one of its children. $P(pg_i|\theta, \hat{\mathcal{G}})$ is the prior probability distribution of parse graph pg_i given $\mathcal{G}$. To simply the learning process, we set it as a uniform distribution. Thus,

$$P(\theta|\mathcal{D}, \hat{\mathcal{G}}) \propto \prod_{D_i \in \mathcal{D}} \max_{pg_i} P(D_i|pg_i, \theta, \hat{\mathcal{G}}) \tag{2.6}$$

And,

$$P(D_i|pg_i, \theta, \hat{\mathcal{G}}) = \prod_{v \in V^{AND}} P(Ch_v|v, \theta_v^{AND}) \tag{2.7}$$

$$\prod_{v \in V^{OR}} P(Ch_v|v, \theta_v^{OR}) \tag{2.8}$$

$$\prod_{v \in V^{T}} P(D_i|v) \tag{2.9}$$

where Ch_v denotes the child of a non-terminal node $v \in V^{AND} \cup V^{OR}$. The probability derivation represents a generating process from a parent node to its child node, and stops at the terminal nodes to generate the sample D_i. The parameters are learned in an iterative process through a Minimax Entropy algorithm explained in more detail later.

As defined by Tu et al. [1], the energy of a parse graph is determined by the energy of the Or-node selection and relations between the And-node children. We define the energy term as $E_{STC}(pg) = E_S(pg) + E_T(pg) + E_C(pg) + \sum E_R(r)$. The energy terms of the model explain why the distribution factorizes as shown in equation 1.

2.3.3 S-AOG: Spatial Concepts Model

A powerful way to capture perceptual information is through a visual grammar to produce the most probable interpretations of observed images. Therefore, we represent spatial concepts through a stochastic Spatial And-Or Graph (S-AOG) [8]. Nodes in the S-AOG represent visual information of varying levels of abstraction. The deeper a node lies in the graph, the more concrete of a concept it represents. An And-node signifies physical compositionality (i.e. a wheel is a part of a car) whereas an Or-node describes structural variation (i.e. a car is a type of vehicle).

As demonstrated in Figure 2.1, the root node of the S-AOG encompasses all possible spatial states a robot may perceive. Here, the "Indoor scene" is decomposed into "Foreground" and "Background," which are then further decomposed. The nodes deeper in the tree represent finer and finer concepts until they end up the terminial nodes consisting of grounded perception units such as the sleeve of t-shirt.

2.3.4 T-AOG: Temporal Concepts Model

The action-space of the world is often an assortment of compositional and variational sub-actions. The hierarchical nature of actions leads us to represent actions by a stochastic Temporal And-Or Graph (T-AOG) [2]. And-nodes correspond to a sequence of actions (i.e. close the door, then lock it), whereas Or-nodes correspond to alternate conflicting actions (i.e. close the door, or open the

door). The leaf nodes of this graph are atomic action primitives that the robot can immediately perform. Different sequences of atomic actions produce different higher-level actions.

The T-AOG structure is learned automatically using techniques from Si et al. [2] establishing an initial knowledge base of actions. Our T-AOG does not learn new atomic actions, but may learn higher-level actions that are built from these atomic actions. By fixing the set of atomic actions, we ensure the grounding of higher-level actions to alleviate the correspondence problem. Our framework assumes detectors of such atomic action as input.

As shown in Figure 2.2, the root node of the T-AOG represents all possible actions. As we traverse the tree down, the actions become less and and less abstract, until they can no longer be simplified. Therefore, the robot can unambiguously perform the atomic actions represented by the leaf nodes.

The T-AOG provides us a way to define the structure and sequence of actions, but how an action causes a change in state is incorporated in the causality data structure defined next.

2.3.5 C-AOG: Causal Concepts Model

Causality is defined as a fluent change due to a relevant action. We can think of fluents as functions on a situation $x_1(s), x_2(s), ...$, such as the state of a car's engine (on vs. off) or its current speed (5mph, 10mph, etc.). We use the Causal And-Or Graph (C-AOG) to encapsulate causality learned from human demonstration [23], as shown in Figure 2.3. Each causal node is a fluent change operator, transforming an input fluent to an output fluent by using an action from the T-AOG. As shown in the diagram, there are various ways to reach the same state. Or-nodes capture the various ways a fluent may change from one state to another.

From the point of view of automated planning, fluents are multi-variate observations of a state. The fluents that change due to a relevant action are vital for predicting future actions. If a fluent does not change from a change-inducing action, then it is irrelevant with respect to the action. These time-invariant properties as defined as "attributes" of the node (i.e. color, weight). Additionally, fluents that change due to an inertial action (i.e. actions that are irrelevant to a fluent change)

are noted inconsistent.

For example, given an cloth s, let fluent $x_1(s)$ represent high-level abstract information such as the shape of a cloth, whereas if the cloth is a shirt, fluent $x_2(s)$ represents specific keypoints for shirts. The C-AOG structure is learned through an information projection pursuit outlined by Fire et al [23]. We assume the observational data correlate strongly to intervention data. The STC-AOG uses these relevant fluent changes to plan out tasks.

2.3.6 Relational Model between Spatial, Temporal, Causal And-Or Graph

Each of the three And-Or Graphs are unified into a common framework for a complete representation of the world [1]. This explicit knowledge is represented by a hierarchical graphical network specifying a stochastic context sensitive grammar [24], called the the Spatial, Temporal, and Causal And-Or Graph (STC-AOG) [1]. The cloth-folding task in our real-time robot framework is incorporated as described in Figure 2.4.

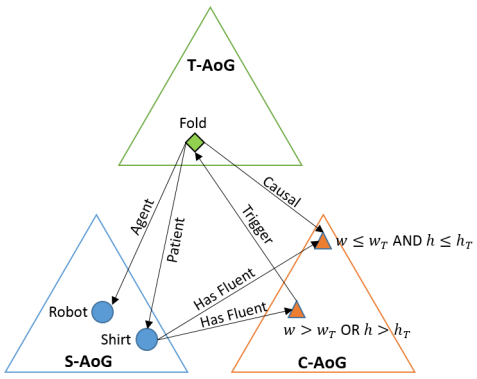


Figure 2.4: For illustrative purposes, this diagram shows simple interactions between the spatial, temporal, and causal And-Or graphs. When the width w or height h of the shirt is larger than the target width w_T or height h_T, the C-AOG triggers a fold action in an attempt to reach a smaller folded shirt. The robot then folds the shirt to produce the desired width and height ($w \leq w_t$ AND $h \leq h_T$).

Formally, the fluent functions $\forall j \; x_i(s_j)$ partition the reals $\mathbb{R}$. Two fluents $x_i(s_a)$ and $x_i(s_b)$ are identical if they belong in the same partition. Each spatial or temporal situation s_i may have

multiple fluents $(x_1, x_2, ...)$.

$$x(s_i) = \begin{pmatrix} x_1(s_i) \\ x_2(s_i) \\ ... \end{pmatrix} \tag{2.10}$$

The fluent change between two states s_j and s_k is formally defined as a binary vector:

$$\triangle x(s_j, s_k) = \begin{pmatrix} \triangle x_1(s_j, s_k) \\ \triangle x_2(s_j, s_k) \\ ... \end{pmatrix} \tag{2.11}$$

$$\triangle x_i(s_j, s_k) = \begin{cases} 0 & \text{if } x_i(s_j) = x_i(s_k) \\ 1 & \text{otherwise} \end{cases}$$

By accumulating human demonstrations of an action, we obtain a set of video clips $Q_a = \{q_1, q_2, ...\}$ for a specific action a, where q_i is a video clip showing action a. The score $w_j(a)$ of an action to make a fluent change is defined as:

$$\forall j \; w_j(a) = P(\triangle x_j = 1 \mid Q) = \frac{\sum_i 1_{\triangle x_j = 1 | q_i}}{||Q||} \tag{2.12}$$

with the scores normalized by $\sqrt{\sum_j w_j(a)^2}$.

Fluents that represent specific properties, such as keypoints, tend to be heavier weighted than those that are broad high-level concepts, such as shape [25]. The fluents are typically hand-chosen, but we suggest automatically generating various abstractions of fluents by varying the dimensionality of autoencoders. Recent work on spatial semantics [26] can also initialize nodes with a set of useful fluents.

The STC-AOG is not just a knowledge representation system, but also a hierarchical planning graph. Folding a shirt using shirt fluents $x_1(s)$ and $x_2(s)$ has greater affordance than that from

using just abstract shape information $x_1(s)$. That way, causal reasoning remains specific to the object, guaranteeing that when folding a shirt, there is less preference to use knowledge about how to fold pants if knowledge about how to fold shirts already exists. We define the affordance of transferring from state s_i to s_j using action a by $\textbf{aff}(a, s_i, s_j) = w(a)^T \triangle x(s_i, s_j)$, suggesting that the automated planning and reasoning should only be based on the relevant features.

Unifying the three sub-graphs produces a closed-loop framework for robots learning from demonstrations. Moreover, graphs can store relationships in an intuitive and highly regular structure, allowing for algorithms that rely on simple graph manipulations. The real world is encoded through perception into the S-AOG to form a physical belief state of the world. The learning algorithm constructs a C-AOG to understand actions from human demonstrations. And lastly, inference combines the reasoning from the C-AOG and the actuators from the T-AOG to physically perform the task. The energy of the joint parse graph [1] combines the energy terms of each:

$$E_{STC}(pg) = E_S(pg) + E_T(pg) + E_C(pg) + \sum_{r \in R^*_{pg}} E_R(r) \tag{2.13}$$

We use generative learning by the Minimax Entropy Principle [27] to learn the probability distribution of STC parse graphs $P(pg)$. Doing so assumes that the sample mean of statistics $\phi_j(pg)$ should approach the true expectation s_j from observations. The parameters are solved by minimizing the Kullback-Leibler divergence between the observed distribution and the candidate $KL(f||p) = E_f[\log f(pg)] - E_f[\log p(pg)]$. This simplifies to a maximum likelihood estimate, formulated by

$$p^* = \underset{p \in \Omega}{\text{argmax}}\ E_f[\log p(pg)] = \underset{p \in \Omega}{\text{argmax}} \sum_{i=1}^{n} \log p(pg_i) + \epsilon \tag{2.14}$$

Iteratively, we choose the statistics $F = \{\phi_1, \phi_2, ...\}$ that minimize the entropy of the model, and the parameters β that yield maximum entropy.

$$p^* = \underset{F}{\text{argmin}}\{\max_{\beta} \text{entropy}(p(pg; \theta)\} \tag{2.15}$$

Effectively, the robot "daydreams" possible probability distributions of parse graphs to converge with observations. During inference, it samples a parse graph to perform the action.

2.3.7 Learning Motor Control

The STC-AOG expresses explicit knowledge in a graphical structure easily understandable by humans, acting as a gateway for communication. However, the STC-AOG only defines discrete salient spatial, temporal, and causal concepts. The interpolation of how an individual action is performed requires a specification of the fine motor skills involved as well as an assignment of probability distribution parameters.

The explicit knowledge captured by a causal node represents a conformant plan learned by human demonstrations. The information stored in the STC-AOG only provides results from discrete time-steps, $t \in \mathbb{N}$. Its state-action table represents fluent changes by $x^{t+1}(s) = f(x^t(s), x^t(a))$. To shift paradigms from explicit to implicit knowledge, we relax the assumption of null run-time observability, and use a finer distinction in time, $x^{t+\delta t}(s) = f(x^t(s), x^t(a))$. By learning this continuous function f, the robot system is capable of verifying, correcting, and inferring causal relations to adapt to dynamic environments.

We make two assumptions to simplify the learning of f. First, we restrict the range of spatial and temporal changes to adhere to spatiotemporal continuity, rendering sudden changes impossible. Second, we use a physical simulator based on perception encoded by the STC parse graph (STC-pg) to compare with reality at rapid time intervals. When a discrepancy is detected, we point fault at the robot's actions. The feedback learning system uses a simplified optimization process inspired by Atkeson et al [28] to update the control mechanics. Adjusting the parameters of the simulator to adhere to reality also reveals useful knowledge, but it is out of scope for this study.

2.3.8 Inference

Since the STC-AOG model is generatively learned, we infer a parse graph through a simple sampling process. As seen in Figure 2.5, the procedurally generated parse graph lays out a conformant

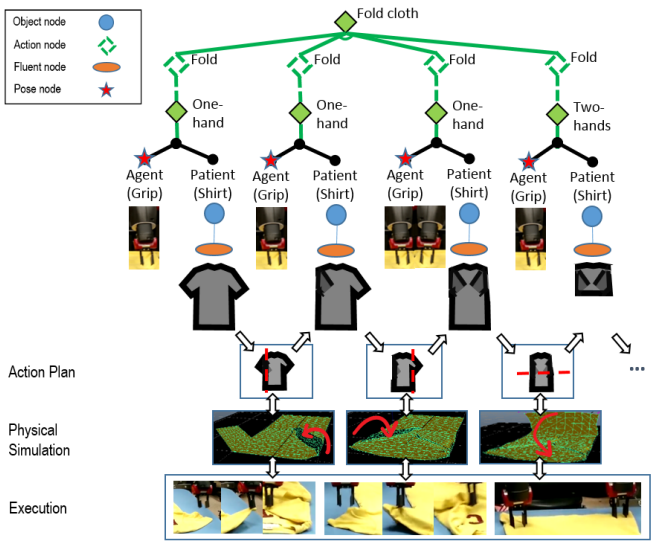


Figure 2.5: The inference engine samples a parse graph to create a conformant action plan. There is feedback between the plan, its simulation, and the corresponding perceived execution.

1: **while** camera is producing image I **do**
2: $pg_S^t \leftarrow \text{Interpret}(G_S, I_t)$
3: $pg_T^t \leftarrow \text{Sample}(G_{STC}, pg_S^t)$
4: $pg_C^t \leftarrow \text{Sample}(G_{STC}, pg_S^t, pg_T^t)$
5: $pg_{STC} \leftarrow \text{Merge}(pg_S^t, pg_T^t, pg_C^t)$
6: PerformWithFeedback(pg_{STC})
7: **end while**

Figure 2.6: The robot inference algorithm performs tasks on a learned STC-AOG. It interprets the sensory input as spatial, temporal, and causal parse graphs, which are merged to formed a joint representation that is sampled and acted on.

action plan for the robot. It then creates a simulation of the action by converting the STC-pg into a motion plan and spatial objects into 3D meshes from point cloud.

The simulation plan is matched with reality at small interval steps to verify that the robot is at its corresponding simulated state. In case of substantial mismatch between expected and actual states, the robot understands the action did not complete, and that a new action plan must be generated based on the latest perception input. Concretely, the sampling procedure is encapsulated by the algorithm in Figure 2.6.

2.4 Experiments

We conduct our experiments on a cloth-folding task. The S-AOG models the physical status of the cloth, table, robot, human, and various decompositions of each. The T-AOG consists of three atomic actions to span the action-space for this simple task: *MoveArm(a)*, *Grab*, and *Release*. Since unsupervised learning on And-Or graphs is not the focus of this chapter, the structure of the S-AOG and T-AOG is fixed, and the parameters are learned. Each graph contains roughly 20 nodes. A *Fold* action in the T-AOG is a higher-level And-node consisting of four children: *MoveArm(a)*, *Grab*, *MoveArm(b)*, and *Release*, with the corresponding textual representation: $Fold(a, b) = MoveArm(a); Grab; MoveArm(b); Release.$ And consequently, a specific instance of folding is a series of *Fold* actions: $FoldStyle1 = Fold(a, b); Fold(c, d); ...; Fold(y, z)$. Lastly, the C-AOG nodes describe how to fold a shirt from one state to another, learned through human demonstrations.

We use Baxter, a two-armed industrial robot to perform our cloth-folding task. Each arm consists of 7 degrees of freedom that are adjusted through inverse kinematics relative to the robot's frame of reference. The robot's primary perception sensor is an Asus PrimeSense camera that provides an aligned RGB-D (Red, Green, Blue, and Depth) point cloud in real-time. In order to use localization results from perception, we compute the affine transformation matrix from the camera coordinate system to that of the robot. All components interact together through the Robot Operating System (ROS).

The STC-AOG is stored in the platform-independent Graphviz DOT language, and used by our platform written in C++. The hand-designed perception logic combines off-the-shelf graph-based [29] and foreground/background [30] segmentation to localize a cloth per frame. On top of that, we train a shirt detector model using a Support Vector Machine to facilitate narrowing down the search for an optimal S-AOG parse graph. Each cloth node has a fluent x_1 describing the low-level shape. If a cloth is a shirt, we represent the structure of its keypoints as another fluent x_2. We simplify learning the probability distribution of parse graphs by limiting the number of statistics to

$F = \{\phi_1\}$, where ϕ_1 is the affordance cost of the action sequence in a STC-pg.

Performance on a task is measured by the percent of successful actions throughout the task. The overall performance is the average of all task performances over multiple trials. An action is successful if performing the action satisfies the pre- and post-conditions of the causal relationship used.

2.4.1 Experiment Settings

In the first set of experiments, we measure the performance of representing learned knowledge from human demonstrations. After watching human demonstrations, the robot generates an action plan step by step. The human performs the action suggested by the robot, and at each step, the human qualitatively verifies whether the robot's action was indeed the intended action as per the demonstration. If verification fails in either case, then the action is marked unsuccessful, and otherwise it is marked successful. This performance score on learning will set the baseline for the next set of experiments.

In the second series of experiments, we measure the quality of grounding the learned knowledge to the robot's actions. This time we let the robot, instead of the human, perform the actions. We compare the performance of the robot folding clothes with the results from the first set of experiments to evaluate the success of grounding physical actions to see how well they match that of a human. The expected performance should be less than the ground truth established from the previous experiment.

In the third series of experiments, we measure the improvements from a feedback system compared to no feedback. We expect that the performance score calculated through this step should be higher than that from the previous experiment, but lower than the ground truth.

Finally, we are also curious how much we can stretch the generalizability of a learned task. After demonstrating how to fold a t-shirt, we ask the robot to infer how to fold different articles of clothing, such as full-sleeve shirts, towels, and pants. The criteria for generalizability of knowledge will follow the similar performance procedure as in the previous experiments.

2.4.2 Results

On 10 trials per four sets of different t-shirt folding demonstrations D_1, D_2, D_3, D_4, we measure the average performance of using our system to learn knowledge, ground robot actions, and control feedback.

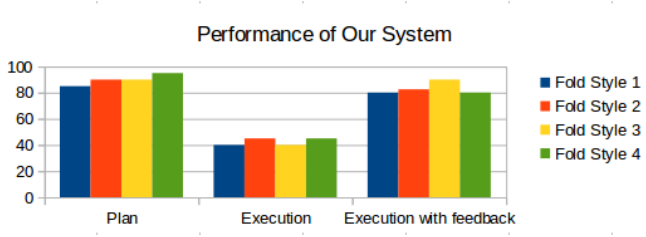


Figure 2.7: Our learning system successfully understood the various folding techniques. It had some difficulty executing the task using simply a conformant plan, but with added feedback the execution was highly successful.

As seen in Figure 2.7, our knowledge representation system was able to characterize the cloth-folding task enough to faithfully communicate with a human, producing a learned representation with an average performance of 90%. This sets the upper bound for the next two inference experiments. As anticipated, our framework was able to ground the actions with a performance of 42.5%. The low score indicates that although the robot knows what to do, there is still a discrepency between the human's action and that generated by the STC-AOG. By adding feedback correction of comparing perception to physical simulation, the performance leaped to 83.125%, also matching our expectation.

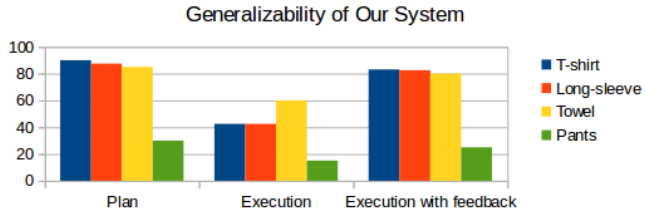


Figure 2.8: Our knowledge framework correctly understood how to generalize a t-shirt folding instruction to long-sleeve shirts and towels; however, it expectedly had difficulty extrapolating its knowledge to fold pants.

The performance of generalizability was measured after training the robot on only t-shirt folding videos. The results are visualized in Figure 2.8. For example, since a full-sleeve shirt may have the same width and height fluents as that of a t-shirt, the inference plan for folding a full-sleeve shirt performed very well. Moreover, the robot was able to generate reasonable action plans to fold a towel it has never seen, since a t-shirt with both its sleeves folded resembles the same rectangular shape of a towel. However, generating a reasonable inference result for folding pants was less successful due to the natural lack of knowledge transferred between a shirt folding and pant folding task. Figure 2.9 shows a few qualitative results of successful folding plans and executions.

Figure 2.9: Some qualitative results on the robot execution after learning from human demonstrations.

2.5 Discussion and Future Work

The experiments show preliminary support for the expressive power of the robot learning and execution framework laid out in this chapter. While we focus heavily in the cloth-folding domain, the framework may be used for training any goal-oriented task. In future work, we wish to continue improving the robustness of each spatial, temporal, and causal And-Or graph to optimize for speed and accuracy.

The STC-AOG acts as a language to ground knowledge and reasoning into robot actions. Since the knowledge representation and robot action planning systems share the same And-Or graph data structure, the graph acts as a programming language for the robot, and self-updating the graph is an act of metaprogramming.

Due to the hierarchical nature of the STC-AOG, the higher level nodes are readily articulated and understandable by humans. are currently working on incorporating natural language state-

ments, commands, and questions to more easily allow humans to manipulate the graph. To scale up the graph for life-long learning, we are investigating other practical storage solutions, including graph-based databases such as Neo4j [31]. Since the graph is sufficient to transfer knowledge, we can upload different skills to a cloud platform and share knowledge between different robots.

Limits in physical reachability and dexterity of the robot arms played a crucial difficulty in mapping action plans to motor control execution. If a grip location was unreachable, the conformant plan would fail to execute the action at all. Fortunately, by introducing the feedback control system, we were able to at least extend the reach as far as possible to grip a reasonable point.

Lastly, the performance of the causal learning system relies on successfully detecting fluent changes. This requires adjusting thresholds for fluent-change detectors until the results seem just right. We solved this problem by offline supervised learning for our chosen fluents, but we set aside the problem of learning these threshold parameters online to future work.

2.6 Conclusions

The stochastic graph-based framework is capable of representing task-oriented knowledge for tractable inference and generalizability. It successfully unified theoretical foundations of And-Or perception grammars to a practical robotics platform. The experimental results support our claims for grounding learned knowledge to execute tasks accurately. We also express the generalizability of our framework by extrapolating from human demonstrations of folding a t-shirt to other articles of clothing. And lastly, our novel framework can make use of perceived discrepancies between high-level action plans and low-level motor control to verify and correct actions.

CHAPTER 3

Situated Dialogue

The last step of the Knowledge Transfer Test is for a robot to propagate knowledge back to a human. Humans communicate through language, so we investigate a way to embed natural language into the STC-AOG structure. Dialogue tends to have some structure: when one says thank you, a common response is you're welcome. We may find clusters in these pairs of utterances, and perhaps we can also cluster those clusters to form higher levels of abstractions in a hierarchical fashion.

3.1 Task Learning through Visual Demonstration and Situated Dialogue

As a new generation of social robots emerges into our daily life, techniques that enable robots to learn task-specific knowledge from human teachers have become increasingly important. In contrast to previous approaches based on Learning from Demonstration [32] and Learning by Instruction [33], we are currently developing a framework that enables task learning through simultaneous visual demonstration and situated dialogue. Supported by our framework, robots can acquire and learn grounded task representations by watching humans perform the task and by communicating with humans through dialogue. The long-term goal is to enable intelligent robots that learn from and collaborate with human partners in a life-long circumstance.

A key element in our framework is And-Or-Graph (AOG) (Tu, Meng, Lee, Choe, and Zhu 2014; Xiong, Shukla, Xiong, and Zhu 2016), which embodies the expressiveness of context sensitive grammars and probabilistic reasoning of graphical models. We use AOG to build a rich representation (i.e., STC-AOG) of the Spatial, Temporal, and Causal knowledge about the real world and the task. In addition, we are also designing an AOG-based schema (i.e., CI-AOG) to model and

interpret the communicative intents between an agent and its human partner. These expressive and deep representations then allow a robot and a human to efficiently and effectively establish and increment their common ground [35] in learning real-world tasks.

This chapter provides an overview of the AOG-based framework and uses an example to illustrate our on-going work on joint task learning from visual demonstration and situated dialogue.

3.2 Representation

3.2.1 STC-AOG

An *And-Or-Graph* (*AOG*) [1] is an extension of a constituency grammar used in Natural Language Processing. It is often visualized as a tree structure consisting of two types of nodes, i.e., *And-node* and *Or-node*. An And-node represents the configuration of a set of sub-entities to form a composite entity; An Or-node represents the set of alternative compositional configurations of an entity. Using this general representation, three important types of task knowledge can be modeled:

- Spatial And-Or Graph (S-AOG) models the spatial decompositions of objects and scenes.
- Temporal And-Or Graph (T-AOG) models the temporal decompositions of events to sub-events and atomic actions.
- Causal And-Or Graph (C-AOG) models the causal decompositions of events and fluent changes.

Robots can utilize the STC-AOG knowledge representation to understand, communicate, and perform task-oriented actions. Based on this knowledge representation framework, Xiong, Shukla, Xiong, and Zhu (2016) has developed a statistical learning mechanism that automatically learns the parameters (e.g., the branching probabilities of Or-Nodes) of S-/T-/C-AOGs from a set of human demonstration videos. Furthermore, methods for learning the structures of different types of AOG have also been studied in previous work (e.g., Pei, Si, Yao, and Zhu 2013; Fire and Zhu 2013).

The basic idea of learning AOG-based task knowledge is to treat each demonstration as a specific instance, or a so-called "parse graph", which is generated by selecting one of the alternative configurations at each Or-node of an AOG model (see Tu, Meng, Lee, Choe, and Zhu (2014) for details). Given a series of demonstrations represented as parse graphs, the structures and parameters of the underlying AOG model then can be learned using statistical learning techniques.

3.2.2 CI-AOG

Since AOG in essence can be viewed as a stochastic grammar machinery, and has been shown powerful in parsing the hierarchical structure of goal-driven events [36], we propose to use the same mechanism for analyzing the intentional structure of knowledge transferring dialogues.

For this purpose, we first construct an AOG, which we call the "Communicative Intent" AOG (CI-AOG) here, to describe how the intentional structure of such dialogues could possibly unfold. Our CI-AOG is similar to the T-AOG or "event grammar" as we illustrated earlier, where an Or-node captures different possibilities and an And-node captures sequential events, and the terminal nodes represent the basic actions (i.e., dialogue acts) that one can perform in a dialogue.

To illustrated the idea, we have manually crafted a (partial) CI-AOG that can be used to analyze the intentional structure of a task teaching dialogue as shown in Figure 3.1. We composed this CI-AOG based on "situated learning" literature [37, 38] to model how the teacher's and the learner's intents interact in a mixed-initiative dialogue. For example, we capture in this CI-AOG the common intentions in situated learning, such as *articulation* (the leaner articulates what is being understood regarding the current situation), *reflection* (the learner reflects what has been learned), and *assessment* (the teacher provides feedback to the learner's reflections or articulations).

Furthermore, the CI-AOG is also used to capture the unique characteristics of dialogue, including turn-taking, initiatives, and collaborative dynamics [35, 39]. To capture the turn-taking dynamics in dialogue, each node in CI-AOG is assigned a role (i.e., who the speaker is). This is illustrated in Figure 3.1 by assigning different colors to the nodes (i.e., orange nodes represent the learner and blue nodes represent the teacher). Therefore, an And-node in CI-AOG not only

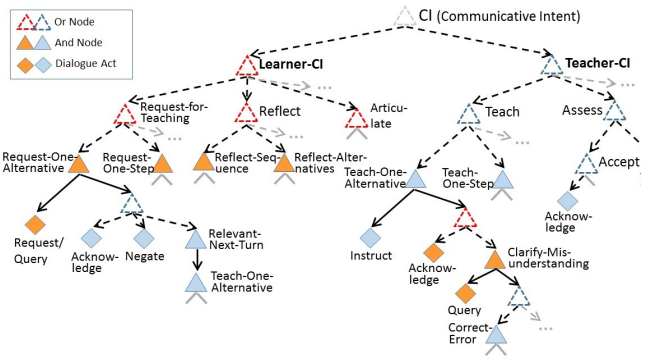


Figure 3.1: An example of Communicative Intent AOG (CI-AOG).

represents the temporal order of its children nodes, but also captures who takes the initiative of the sub-dialogue and how the turn-taking switches between the learner and the teacher.

The expressiveness of the AOG language also allows us to capture the collaborative dynamics studied in the discourse analysis literature (e.g., Clark and Schaefer (1989)). For example, as illustrated in the left bottom part of Figure 3.1, after the learner requests the teacher for teaching an alternative way of doing a task (i.e., the *Request-One-Alternative* node), the teacher should respond an explicit acknowledgement, or a negation, or directly teach an alternative without explicit acknowledging (the "relevant-next-turn" behavior).

Suppose a CI-AOG has already been constructed, it then can be used for "parsing" the underlying intentional structure of an ongoing dialogue. This is similar to previous work [36] that used a Top-Down parsing algorithm to analyze the hierarchical structure of goal-driven events from an observed sequence of atomic actions. Figure 3.2 further illustrates a parse graph on the underlying intentional structure of the following example dialogue.

Example dialogue of a robot learning to fold a t-shirt:

R1: Could you teach me how to fold the t-shirt?

H1: Sure.

H2: First, you fold the right sleeve towards the middle of the t-shirt.

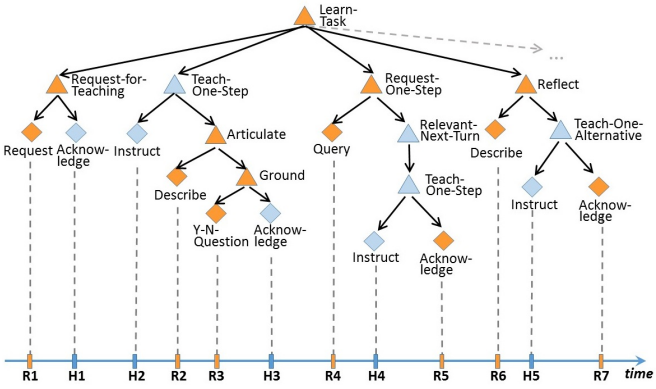


Figure 3.2: The CI parse graph for the given example dialogue.

R2: I saw you approached the right sleeve, grasped there, then moved to a position, and released your hand there.

R3: Is that position the middle of the t-shirt?

H3: Yes, that position is in the middle of the t-shirt.

R4: OK, what is the next step?

H4: Next, you fold the left sleeve to the middle.

R5: OK.

R6: This is what I have learned: first fold the right sleeve to the middle, and then fold the left sleeve to the middle.

H5: You can also fold the left sleeve first, and then the right sleeve.

R7: I see.

R8: What is next?

. . . . . .

As illustrated in Figure 3.2, the overall intent of this dialogue is for the robot to learn a sequential task. It is initiated by the robot's request for teaching (R1), followed by the human's explicit acknowledgement (H1). The following sub-dialogue is then led by the human's intent of

teaching the robot the first step with an instruction (H2). Following that, the robot articulates what it understands about the current situation (R2), and tries to map the unknown concept "middle" to a physical position in the visual context (the question asked in R3, with an intent of what we call "*ground*"). The human's positive response (H3) confirms the robot's understanding, and also closes the subroutine of teaching the first step. The dialogue routine then rolls back to a higher-level of the intent hierarchy, where the robot moves on with its intent of learning the next step (R4). In R6, after two consecutive steps have been learned, the robot issues a reflection on what has been learned so far, which triggers human's following intent to teach an alternative order (H5).

Now we have introduced different types of AOG as the fundamental representations of the physical world, task knowledge, and dialogue dynamics. Next we turn our focus to discussing how we utilize these representations to build learning agents under a unified framework.

3.3 Learning from Situated Dialogue

Natural language and dialogue can play an important role in learning task knowledge from a human. Language provides a key source of information to gear the learned knowledge towards how humans conceptualize and communicate about situations and tasks. Such "human-oriented" knowledge is very necessary for facilitating human-robot communication and collaboration (for example, Lemon, Gruenstein, and Peters (2002)).

Furthermore, dialogue provides an expedited way to learn task knowledge. This can be demonstrated by our earlier example of learning how to fold a t-shirt. After the robot reflected (in R6) the just learned two steps (i.e., *fold-right-sleeve* and *fold-left-sleeve*), the human further taught that the order of the two steps could be switched and it would result into the same status of performing the task (H5). With our AOG-based representation, the robot can add this new knowledge by directly modifying the high-level structure of the STC-AOG (i.e., create new temporal and causal Or-Nodes to represent this alternative sequence of actions and fluent changes). Using language makes it much easier to communicate such high-level knowledge (Figure 3.3 illustrates the STC-

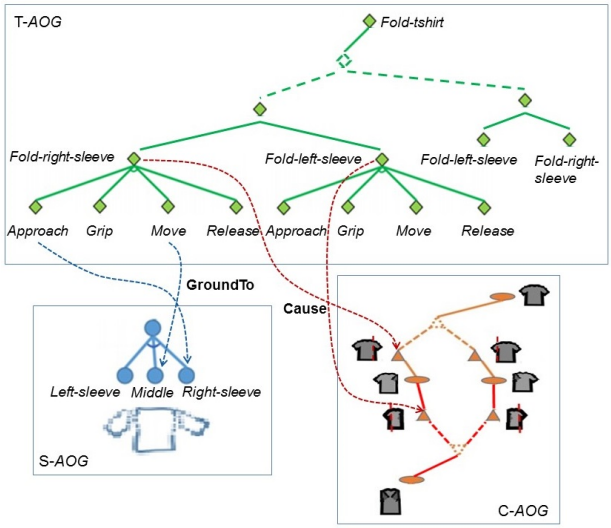


Figure 3.3: The STC-AOG representation of task knowledge that can be learned from the previous example dialogue of learning to fold a t-shirt. Note that the S-/T-/C- components are not independent from each other. The interplay between them provides an integrated representation of the task knowledge.

AOG representation that can be learned thereafter).

We thus propose an AOG-based framework to enable robot learning task knowledge from natural language and visual demonstration simultaneously. Supported by this framework, the robot can also proactively engage in human's teaching through dialogue, and gradually accumulate and refine its knowledge. One key advantage of our proposed framework is to provide a unified view of modeling the joint and dynamic task learning process. Besides, since we use AOG as a common representation basis, different components of our model can be stored and accessed using the same format (e.g., graph database), and be processed by the same set of algorithms. It thus can greatly ease the burden of building complex AI agents.

Figure 3.4 illustrates the basic ideas of our task learning system. It mainly consists of three tightly connected components that are all based on AOG representation and processing:

- *Language and Vision Understanding* processes the visual context into a "Vision Parse Graph" (*V-PG*) and the linguistic context into a "Language Parse Graph" (*L-PG*), and fuses them together into a "Joint Parse Graph" (*Joint-PG*) for a deep and accurate understanding of the

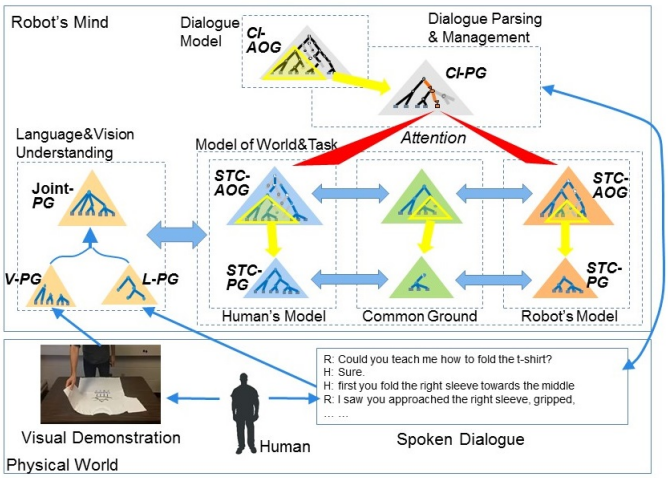


Figure 3.4: Illustration of our AOG-based framework for supporting robot learning from situated dialogue.

current situation. A previous work [1] has employed the same AOG-based representations for joint text and video parsing in the question-answering domain. The processing in our component here resembles that work. However the linguistic content of a dialogue could require more sophisticated approaches than those for handling monologues, and our goal is to learn generalizable task knowledge rather than just understand one situation.

- *World and Task Model* manages the representation and acquisition of knowledge of the physical world and tasks. As introduced earlier, we use STC-AOG to represent general knowledge about the world and the tasks, while a specific situation (i.e., a Joint Parse Graph) is represented as an instantiation (or sub-graph) of the STC-AOG. Motivated by the Common Ground theory [35], our agent maintains three copies of models. One is the human's model of the world and knowledge, which is inferred from the joint parsing of language and vision. One is the agent's own model, and the third one is their shared/matched understanding of the situation and knowledge of the task (i.e., their *common ground*). In future work, we will further extend these models towards modeling the "Theory of Mind" in human-robot collaboration.

- *Dialogue Modeling and Management* uses CI-AOG to model and analyze the intentional

structure of the task learning dialogue, and to facilitate the agent's decision making in knowledge acquisition and dialogue engagement. Our design of the situated dialogue agent also resembles the classical theory on discourse modeling [41]. I.e., the *intentional structure* is captured by a CI- Parse Graph (CI-PG) in our dialogue management component. The *linguistic structure* in our case has been extended to the joint model of the linguistic and visual contexts (captured as STC- Parse Graphs), and the shared knowledge (captured as STC-AOG). The *attentional state* is captured by linking each node in the CI-PG to a specific node/edge in the situation or knowledge representation graphs.

As the dialogue and demonstration unfold, the agent dynamically updates its intent, situation, and knowledge graphs. Each component can utilize the information from others through the interconnections between their graph representations. Based on this unified framework, sophisticated learning agents can become easier to be designed and built.

3.4 Conclusion

This chapter provides a brief overview of our on-going investigation on integrating language, vision, and situated dialogue for robot tasking learning based on And-Or-Graphs (AOG). In particular, through an example, it demonstrates how language and dialogue can be used to augment visual demonstration by incorporating higher-level knowledge. Here we use cloth-folding as an example, but the same framework can be extended to other types of task learning. We are currently in the process of implementing the end-to-end system and plan to collect realistic data to evaluate our approach.

CHAPTER 4

Human Utility

One condition to pass the Knowledge Transfer Test is that a robot must score sufficiently well on performing the intended task. If we train a robot on videos of people folding t-shirts, then evaluating the robot's performance on folding the very same t-shirt may not be indicative of grasping the more general concept of arbitrary cloth-folding.

The STC-AOG up until now has modeled a task only by memorizing action trajectories and state-changes from human demonstrations. We now shift our focus to learning human utility, so that the robot can model the task of folding general articles of clothing from just a few videos of people folding t-shirts. The complete representation of a task is therefore both the STC-AOG, which captures what fluents change based on what actions are taken, and the utility model, which captures the value of changing fluents. The robot is driven by the utility landscape, in a non-Markovian planning framework we call fluent dynamics.

4.1 Learning Human Utility from Video Demonstrations

Explicitly programming service robots to accomplish new tasks in uncontrolled environments is time-consuming, error-prone, and sometimes even infeasible. In Learning from Demonstration (LfD), many statistical models have been proposed that maximize the likelihood of observations [32]. For example, Bayesian formulations [42] assume a prior model of the goal, and use Bayes' Theorem to explain the relationship between the posterior and likelihood. These Bayesian formulations learn a model of the demonstrated task most consistent with training data. Such approaches are often referred to as *inductive learning* [43].

In contrast, robot autonomy was originally studied as a rule-based deductive learning system

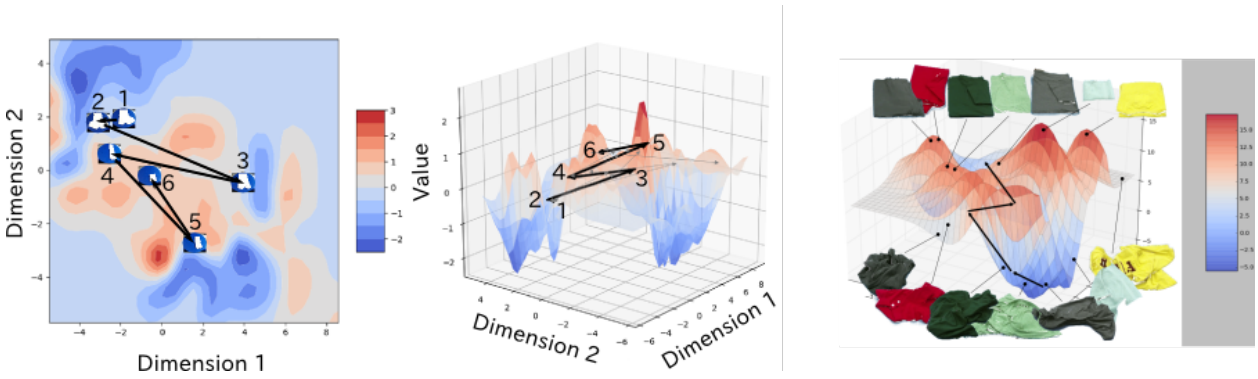


Figure 4.1: The utility landscape identifies desired states. This one, in particular, is trained from 45 cloth-folding video demonstrations. For visualization purposes, we reduce the state-space to two dimensions through multidimensional scaling (MDS). The canyons in this landscape represent wrinkled clothes, whereas the peaks represent well-folded clothes. Given this learned utility function, a robot chooses from an available set of actions to craft a motion trajectory that maximizes its utility.

[44, 45]. There is a paradigm shift in applying inductive models to *deduction* based inference. In this chapter, we explore a middle-ground, where deductive rules are learned through statistical techniques.

Specifically, we teach a robot how to fold shirts through human demonstrations, and have it reproduce the skill under both different articles of clothing and different sets of available actions. Our experimental results show good performance on a two-armed industrial robot following causal chains that maximize a learned latent utility function. Most importantly, the robot's decisions are interpretable, facilitating immediate natural language description of plans [13].

Human preferences are modeled by a latent utility function over the states of the world. To rank preferences, we pursue relevant fluents of a task, and then learn a utility function based on these fluents. For example, Figure 5.1 shows the utility landscape for a cloth-folding task, obtained through 45 visual demonstrations.

The utility landscape shows a global perspective of candidate goal states. To close the loop with autonomous behaviour, we further design a dynamics equation to connect high-level reasoning to low-level motion control. The primary contributions of our work include:

1. Learning an *interpretable* utility of continuous states, independent of system dynamics.

2. *Deductively* exploring goal-reachability under different available actions.

3. Proposing *"Fluent Dynamics"* to bridge low-level motion trajectory with high-level utility.

4. Teaching a robot to fold t-shirts, and have it *generalize* to arbitrary articles of clothing.

4.2 Related Work

Modeling and Learning Human Utility: Building computational models for human utilities could be traced back to the English philosopher, Jeremy Bentham, in his works on ethics known as utilitarianism [46]. Utilities, or values, are also used in planning schemes like Markov decision process (MDP) [47], and are often associated with states of a task. However, in the literature of MDP, the "value" is not a reflection of true human preference and, inconveniently, is tightly dependent on the agent's actions.

Zhu, Jiang, Zhao, Terzopoulos, and Zhu [48] first modeled human utilities over physical forces on tools, and proposed effective algorithms to learn utilities from videos. Our work differs in 4 ways: 1) we generalize the linear SVM separator ("rankSVM") so that the utility of each individual fluent is dictated not by a "weight" but instead a non-linear utility function; 2) relevant fluents are pursued one-by-one from a large dictionary; 3) we learn from external fluents, such as states of the world, instead of internal fluents, such as states of the agent; 4) the utility function drives robot motion.

Inverse Reinforcement Learning: Inverse reinforcement learning (IRL) aims to determine the reward function being locally optimized from observed behaviors of the actors [49]. In the IRL framework, we are satisfied when a robot mimics observed action sequences, maximizing the likelihood. Hadfieldmenell, Dragan, Abbeel, and Russell [50] defined cooperative inverse reinforcement learning (CIRL), which allows reward learning when the observed behaviors could be sub-optimal, based on human-robot interactions. In contrast to IRL or CIRL, our method does

not aim to reproduce action sequences, nor is our approach dependent on the set of possible actions. It avoids the correspondence problem in human-robot knowledge transfer by learning the global utility function over observed states, rather than learning the local reward function from actions directly. Furthermore, our work avoids the troublesome limitations of subscribing to a Markov assumption [51, 52].

Robot Learning from Demonstrations: Learning how to perform a task from human demonstrations has been a challenging problem for artificial intelligence and robotics, and various methods were developed trying to solve the problem [53]. Learning the task of cloth-folding from human demonstrations, in particular, has been studied before by Xiong, Shukla, Xiong, and Zhu [34]. While most of the existing approaches focus on reproducing the demonstrator's action sequence, our work tries to model human utilities from observations, and generates task plans deductively from utilities.

4.3 Model

Definition 1. Environment: The world (or *environment*) is defined by a generative composition model of objects, actions, and changes in conditions [14]. Specifically, we use the stochastic context-free And-Or graph (AOG), which explicitly models variations and compositions of spatial (S), temporal (T), and causal (C) concepts, called the STC-AOG [34].

The atomic (*terminal*) units of this composition grammar are tuples of the form $(F_{start}, u_{[1:t]}, F_{end})$, where F_{start} and F_{end} are pre- and post-fluents of a sequence of interactions $u_{[1:t]}$. Concretely, the sequence of interactions $u_{[1:t]}$ is implemented by spatial and temporal features of human-object interactions (4D HOI) [54]. See definition 9.

Definition 2. State: A *state* is a configuration of the believed model of the world. In our case, a state is a parse-graph (pg) of the And-Or graph, representing a selection of parameters (Θ_{OR}) for each Or-node. The set of all parse-graphs is denoted Ω_{pg}.

Definition 3. Fluent: A *fluent* is a condition of a state that can change over time [55]. It is

represented as a real-valued function on the state (indexed by $i \in \mathbb{N}$): $f_i : \Omega_{pg} \rightarrow \mathbb{R}$.

Definition 4. Fluent-vector: A *fluent-vector* F is a column-vector of fluents: $F = (f_1, f_2, ..., f_k)^\intercal$.

Definition 5. Goal: The *goal* of a task is characterized by a fluent-change $\triangle F$. The purpose of learning the utility function is to identify reasonable goals.

4.3.1 Utility Model

We assume human preferences are derived from a utilitarian model, in which a latent utility function assigns a real-number to each configuration of the world. For example, if a state pg^1 has a higher utility than another state pg^2, then the corresponding ranking is denoted $pg^1 \succ pg^2$, implying the utility of pg^1 is greater than the utility of pg^2.

Each video demonstration contains a sequence of n states $pg^0, pg^1, ..., pg^n$, which offers $\binom{n}{2} = n(n-1)/2$ possible ordered pairs (*ranking constraints*). Given some ranking constraints, we define an energy function by how consistent a utility function is with the constraints.

The energy function described above is used to design its corresponding Gibbs distribution. In the case of Zhu and Mumford [56], a maximum entropy model reproduces the marginal distributions of fluents. Instead of matching statistics of observations, our work attempts to model human preferences. We instead use a maximum margin formulation, and select relevant fluents by minimizing the ranking violations of the model. The specific details of this preference model is described below.

4.3.2 Minimum Violations

Let $\mathcal{D} = \{f^{(1)}, f^{(2)}, ...\}$ be a dictionary of fluents, each with a latent utility function $\lambda : \mathbb{R} \rightarrow \mathbb{R}$. Using a sparse coding model, the utility of a parse-graph pg is estimated by a small subset of relevant fluents $F = \{f^{(1)}, f^{(2)}, ..., f^{(K)}\} \subset \mathcal{D}$. Denote $\Lambda = \{\lambda^{(1)}(), \lambda^{(2)}(), ..., \lambda^{(K)}()\}$ as the corresponding set of utility functions for each fluent in F. For example, 12 utility functions learned from human preferences are shown in Figure 5.3, approximated by piecewise linear functions. The

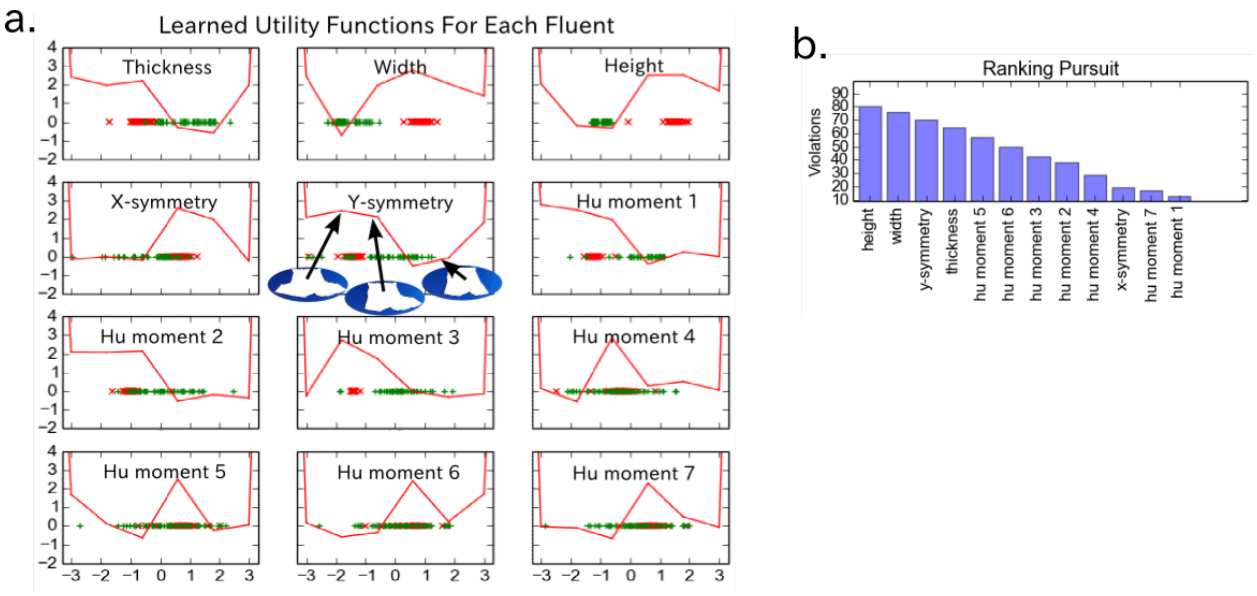


Figure 4.2: **(a)** The 12 curves represent the negative utility function $(-\lambda)$ corresponding to each fluent. The functions are negated to draw parallels with the concept of potential energy. Red marks indicate fluent values of pg^0, which the learned model appears to avoid, and the green marks indicate fluent values of the goal pg^*, which the learned model appears to favor. Notice how the y-symmetry potential energy decreases as the cloth becomes more and more symmetric. By tracing the change in utilities of each individual fluent, the robot can more clearly explain *why* it favors one state over another. **(b)** The ranking pursuit algorithm extracts fluents greedily to minimize ranking violations. As shown in the chart, the top 3 most important fluents for the task of cloth-folding are height, width, and y-symmetry.

total utility function is thus,

$$U(pg; \Lambda, F) = \sum_{\alpha=1}^{K} \lambda^{(\alpha)}(f^{(\alpha)}(pg)) \tag{4.1}$$

Of all selection of parameters (Λ) and fluent-vectors (F) that satisfy the ranking constraints, we choose the model with minimum ranking violations. In order to learn each utility function in Λ, we treat the space of fluents as a set of *alternatives* [57]. Let R denote the set of rankings over the alternatives. Each human demonstration is seen as a ranking $\sigma_i \in R$ over the alternatives. We say $a \succ_{\sigma_i} b$ if person i prefers alternative a to alternative b. The collection of a person's rankings is called their preference profile, denoted $\vec{\sigma}$.

Each video v provides a preference profile $\vec{\sigma}_v$. For example, we assume at least the following ranking: $pg^* \succ_{\sigma_v} pg^0$, where pg^0 is the initial state and pg^* is the final state. The learned utility functions try to satisfy $U(pg^*) > U(pg^0)$.

U is treated as a ranking score: higher values correspond to more favorable states. We want to model the goal of a task using rankings obtained from visual demonstrations. The goal model, or *preference model*, of a parse-graph pg takes the Gibbs distribution of the form, $\mathbf{p}(pg; \Lambda, F) = \frac{1}{Z}\mathbf{e}^{U(pg;\Lambda,F)}$, where $U(\ \cdot\ ; \Lambda, F)$ is the total utility function that minimizes ranking violations:

$$\begin{aligned} \min \sum_{\alpha=1}^{K} \int_x \lambda''^{(\alpha)} dx + C \sum_v \xi_v \\ \text{s.t. } \sum_\alpha \left(\lambda^{(\alpha)}(f^{(\alpha)}(pg_v^*)) - \lambda^{(\alpha)}(f^{(\alpha)}(pg_v^0))\right) \\ > 1 - \xi_v, \\ \xi_v \geq 0. \end{aligned} \tag{4.2}$$

Here, ξ_v is a non-negative slack variable analogous to margin maximization. C is a hyper-parameter that balances the violations against smoothness of the utility functions [58]. Of all utility functions, we select the one which minimizes the ranking violations. The next section explains how to select

the optimal subset of fluents.

4.3.3 Ranking pursuit

The empirical rankings of states $pg^* \succ pg^0$ in the observations must match the predicted ranking. We start with an empty set of fluents $F = \{\}$, and select from the elements of $\mathcal{D}$ that result in the least number of ranking violations.

This process continues greedily until the amount of violations can no longer be substantially reduced. Figure 5.3 shows empirical results of pursuing relevant fluents for the cloth-folding task. The dictionary of initial fluents may be hand-designed or automatically learned through statistical means, such as from hidden layers of a convolutonal neural network.

4.3.4 Ranking Sparsity

The number of ranking pairs we can extract from the training dataset is not immediately obvious. For example, each video demonstration supplies ordered pairs of states that we can use to learn a utility function. A sequence of n states (pg^0, pg^1, ..., pg^n) allows $\binom{n}{2} = n(n-1)/2$ ordered pairs.

On one end of the spectrum, which we call *sparse ranking*, we know at the very least that $pg^n \succ pg^0$ for each demonstration. This is a safe bet since each video demonstration is assumed to successfully accomplish the goal. However, the utility model throws out useful information when ignoring the intermediate states.

On the other end, in *dense ranking*, all $\binom{n}{2}$ are used. Despite using all information available, this approach may be prone to introducing many ranking violations.

Figure 4.3 visualizes performance of both approaches as we increment the number of available video demonstrations.

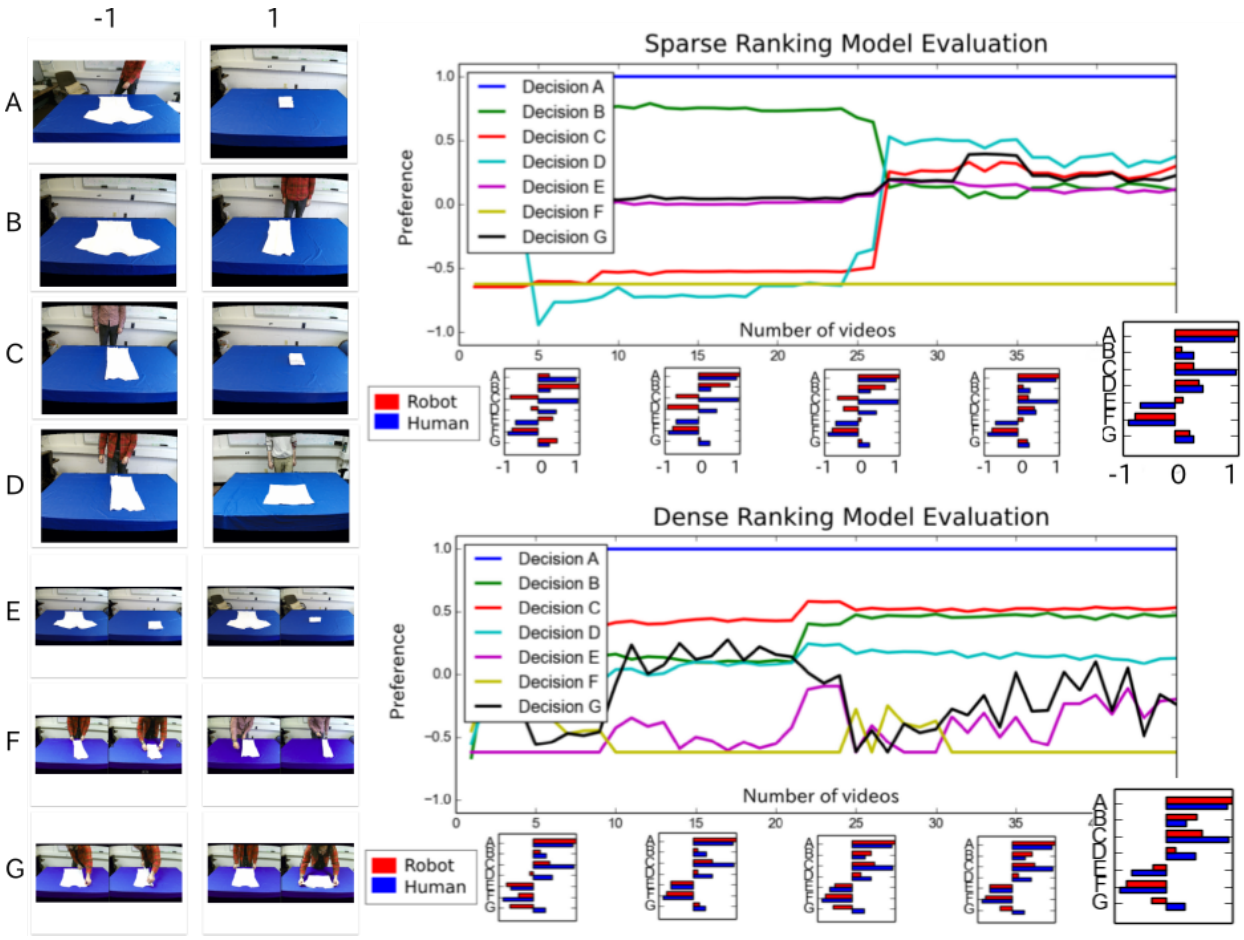


Figure 4.3: The sparse and dense ranking models are evaluated by how quickly they converge and how strongly they match human preferences. The x-axis on each plot indicates the number of unique videos shown to the learning algorithm. The y-axis indicates two alternatives (1 vs. -1) for 7 decisions (A, B, C, D, E, F, and G) of varying difficulty. The horizontal bar-charts below each plot show comparisons between human and robot preferences. As more videos are made available, both models improve performance in convergence as well as alignment to human preferences (from 330 survey results).

4.4 Utility-Driven Task Planning

The learned utility function is used to drive robot behavior. In order to perform low-level motion control in the real world, the robot needs a concept of what actions are possible in the observed state. First, we explicitly define the concept of relevant fluents, and then use it to describe the preconditions of an action.

Definition 6. Relevant fluent: A fluent-vector might contain irrelevant fluents for an action. The *relevant fluents of an action* are a subset of a fluent-vector, described using an element-wise multiplication of a binary vector w by the fluent-vector, $w \circ F = WF$, where W is a diagonal matrix.

Definition 7. Action: An *action* sequence $u_{[1:t]}$ causes a change in fluents given a precondition. The precondition of an action depends on relevant fluents W_u as well as a representative example pg_u. Let $\theta = (W_u, pg_u)$ denote these parameters. The precondition is a probability $P(X = pg \mid u;\ \theta_u)$ over a random variable $X \in \Omega_{pg}$.

A tuple of (F_{start}, $u_{[1:t]}$, F_{end}) is used to construct the compositional model of the environment. The energy of a pg is computed by comparing the difference between the relevant fluents of the observation Wpg to the relevant fluents of the action Wpg_u, Cost$(pg;\ W_u, pg_u) = ||W_u(pg - pg_u)||$.

4.4.1 Representing what, how, and why

Fluents provide information on the utility of the state. Therefore, a robot can identify a fluent-change $\triangle F$ to maximize its utility, explaining *why* it acts. Figuring out how to achieve the fluent-change requires the robot to accomplish a sequence of interactions between itself and the environment, which we call the *union space*. Interacting with the environment requires the robot to be aware of the span of its own actuators, called the *actuator space*, where it identifies *what* joints move.

The three spaces are tightly coupled, and can be used to perform real-time robot executions. We explain each space separately, and then provide a unified dynamics formulation for robot task planning.

Definition 8. Actuator Space: We characterize a robot actuator by its η degrees-of-freedom. The actuator space Ω_A is a set of all valid η-dimensional vectors. At any point in time, a robot can be represented as a point in this space, $a \in \Omega_A \subset \mathbb{R}^\eta$.

For example, our robot platform is represented by a 16-dimensional vector, since each arm has 7 joints and an open/close grasp-status for each hand. As the robot moves in the real-world, this 16-dimensional point drifts in the actuator space.

Definition 9. Union Space: The interactions between an agent and object are jointly interpreted in what we call the union space Ω_U. The distance between an agent's end-effector and the midpoint of an object is one such example. These computed values depend of the current actuator position, $u(a) \in \Omega_U$. The table below shows a possible 12-dimensional vector in the union space.

#	**Feature**	**Agent**	**Object**
1	distance	left end-effector	cloth
2-5	θ_x, θ_y, θ_z, θ_w	left end-effector	cloth
6	grasp status	left end-effector	N/A
7	distance	right end-effector	cloth
8-11	θ_x, θ_y, θ_z, θ_w	right end-effector	cloth
12	grasp status	right end-effector	N/A

Examples of rows in this table include, but are not limited to, distance and orientation between an end-effector and an object. Clearly, a robot adjusting its position in the actuator space affects its position in the union space. A sequence of such vectors in the union space causes a fluent change. Inferring how fluents change from a trajectory in the union space is given as a hierarchical task plan (see Definition 1) by the domain expert. Further causal relationships are gathered through exploration, but a deeper investigation in learning causality is beyond the scope of this chapter.

Definition 10. Fluent Space: A fluent space Ω_F is the set of all possible fluent vectors F. A sequence of vectors from the union space $u_{[1:t]}$ causes the value of the fluent vector to change, $F(u_{[1:t]}) \in \Omega_F$.

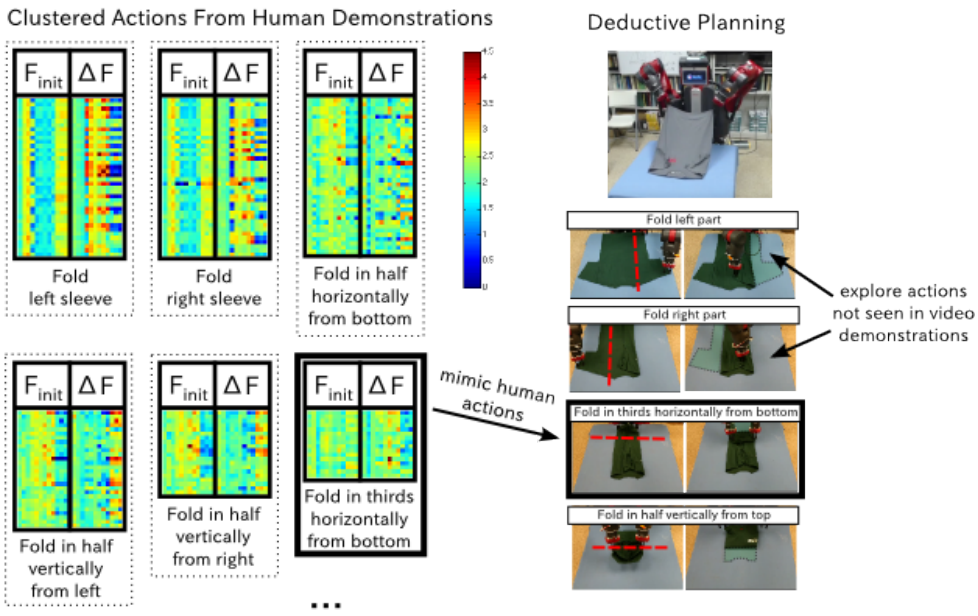


Figure 4.4: After clustering fluent-changes from the training videos, 11 actions are automatically captured. Each action is shown by 2 matrices: F_{init} and $\triangle F$. The rows of the matrix correspond to a concrete example. The columns of the matrix correspond to the various fluents. The robot can understand the relevant fluents associated per each action.

4.4.2 Fluent Dynamics

Influenced by previous work [59] in specifying robot behavior independently of its actuators, this chapter proposes a control formulation to unify low-level mechanics with high-level preference. In deductive planning, the optimal selection of actions achieves a goal with minimum cost. We define the cost of a sequence of actions $V(a_{[1:t]})$ by the utility of the resulting fluent-vector. With that in mind, the robot must use its available actionable information to maximize utility, $a^*_{[1:t]} = \arg\max_{a_{[1:t]}} V(a_{[1:t]})$. Optimal action sequences will satisfy $\partial V / \partial a_{[1:t]} = 0$. The gradient of V with respect to $a_{[1:t]}$ can be computed using the chain rule,

$$\frac{\partial V}{\partial a_{[1:t]}} = \frac{\partial V}{\partial F} \frac{\partial F}{\partial u_{[1:t]}} \frac{\partial u_{[1:t]}}{\partial a_{[1:t]}} \tag{4.3}$$

The first factor $\partial V / \partial F$ comes immediately from the utility learning section of this chapter. The second factor $\partial F / \partial u_{[1:t]}$ is solved using the assumed And-Or compositional model of the world, as explain in Definition 1. And lastly, inverse kinematics and optimal control methods directly solve $\partial u_{[1:t]} / \partial a_{[1:t]}$ [59].

The trajectory Γ between two robot states is $\arg\min_{a_{[1:t]}} \sum_{a_{[1:t]}} |V(a_{[1:t]}) \cdot \triangle a_{[1:t]}|$ [60].

The optimal trajectory of actions to achieve the highest value is estimated using beam-search and dynamic programming. Sub-goal reachability is automatically solved through the *Confident Execution* algorithm in [61] as shown in Figure 4.4.

4.5 Implementation and Experimental Results

We learn our utility function from 45 RGB-D (*pointcloud*) video demonstrations of t-shirt folding. This dataset splits into 30 for training and 15 for testing. The videos were recorded on a separate Kinect camera at a different orientation in a separate setting by different people.

At each frame, the vision processing step segments the cloth using graph-based techniques on the 2D RGB image [29, 30], and then the extracted 3D cloth pointcloud is aligned to its principal

axis. Next, we extract fluents from the pointcloud being tracked. Examples of a couple fluents include width, height, thickness, x-symmetry, y-symmetry, and the 7 moment invariants [62]. Width, height, and thickness are calculated in meters, after aligning the segmented pointcloud to its principal axis. X-, and y-symmetry scores are calculated by measuring the symmetric difference of folding the segmented pointcloud on the first two principle axes. The Hu-moments are calculated on the binary mask of the cloth.

We extract both automatic and hand-designed fluents to obtain a training dataset $\mathcal{X}$ to pursue relevant fluents and obtain an optimal ranking. Each utility function is approximated by a piecewise linear function. The vision processing code, ranking pursuit algorithm, and the RGB-D dataset of cloth-folding are open-sourced on the author's website [1].

We cross-validate the fitting of the learned utility model to ensure generalizability to the other training videos. Furthermore, we perform external evaluations on 330 individuals to compare how well the learned preference model matches human judgement. In this survey, each human was asked to make a decision on 7 choices after being told, "A robot attempts to fold your clothes. Of each outcome, which do you prefer?"

The qualitative comparisons between human and robot preferences is shown in figure 4.3. Overall, our model quickly converges to human preferences (within 27 videos in the sparse ranking model and just 5 videos in the dense ranking model).

To further evaluate how effectively the learned utility function assists in deductive planning, we conduct a series of experiments on a robot platform. We solve the three factors in the dynamics equation independently. The learned utility function gives us the first factor $\partial V/\partial F$, the STC-AOG units provide the second factor $\partial F/\partial a_{u_{[1:t]}}$, and an off-the-shelf inverse kinematics library solves $\partial u_{[1:t]}/\partial a_{[1:t]}$.

In the experiments, the robot is presented with a never-before-seen article of clothing, and asked to identify a goal, plan an action sequence to achieve that goal, predict the utility gain, and then perform the plan to compare against actual utility gain. Figure 4.5 shows how well the robot's

[1] `https://github.com/BinRoot/Fluent-Extractor`

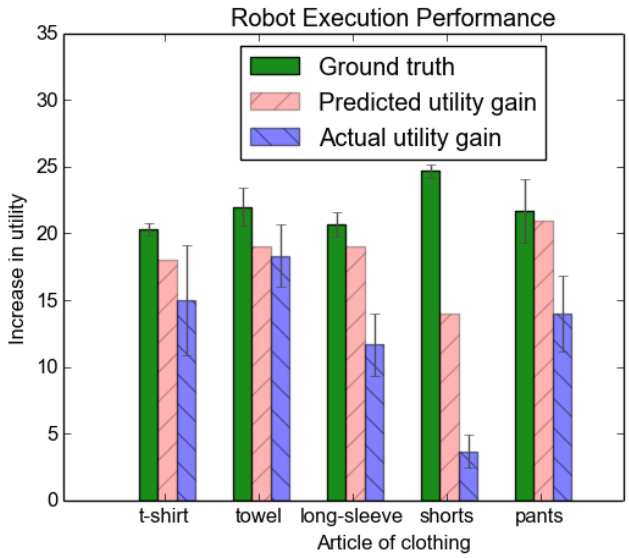


Figure 4.5: Robot task execution is evaluated in two ways. First, we measure how well the robot can predict the increase in utility through deductive reasoning compared to ground truth by having a human perform the same action. Second, we compare the prediction to actual utility gain after executing the action sequence using Fluent Dynamics. Our experiments show strong generalizability to other articles of clothing, even though the training videos only contained t-shirt folding demonstrations. As shown above, the robot can detect execution failure when folding shorts by detection an anomaly in actual vs. predicted utility gain.

performance matches both its own predictions as well as the ground truth.

4.6 Conclusions and Future Work

The learned utility model strongly matches preferences of 330 human decisions, capturing the commonsense goal. The inferred action plan is not only interpretable (*why*, *how*, and *what*), but also performable on a robot platform to complete the learned task in a completely new situation. In future work, we would like to incorporate social choice theory [63], understand inconsistencies between human preferences [64], and learn fluents automatically using neural techniques [65].

CHAPTER 5

Unified Representation of Tasks

If the STC-AOG is additionally given read/write access to fluents, then we can craft a structure that, when run on a robot, can simulate a Turing machine. Consequently, we reinterpret the graphical data-structure of the STC-AOG as a probabilistic programming language, and show how such a language can be used to model tasks in a variety of domains. We also go a step further, and synthesize code in this language from few observations.

We coin the term task-oriented programming language, and show how it can elegantly model tasks by defining an action-space and utility. The AOG data-structure inherent in the abstract syntax tree of a task-oriented programming language can leverage much of the AOG literature (learning, inference, parsing, synthesis, etc.).

5.1 Model

Unlike a Markov Decision Process (MDP) [66, 67], our framework assumes at least a context-sensitive model: the next action depends on both the observed state as well as the unobserved context. While it is possible to regard both the state and context as one hyper-state, doing so defeats the tractability of the Markov assumption. Following situation calculus convention, we regard the aforementioned hyper-state as the *situation* [68], while we regard the currently observable state as the *observation*.

Definition 11 (Observation)**.** Each observation $o \in \Omega_o$ carries some information about sensory signals at that point in space and time. Examples of observations include: (1) there is a wrinkled article of clothing on a table right now from my perspective, (2) the user said the utterance "hello"

just now, and (3) the value of the variable x is currently 3.

Definition 12 (Situation and Action)**.** A situation s is the series of actions $\vec{a}$ taken from an initial situation s_0 [69]. The notation $do(a, s)$ applies an action a to a situation s, resulting in a new situation s'. The composition of actions may be written as $a'' \circ a' \circ a$ or a sequence (a, a', a'').

Definition 13 (Fluent)**.** A fluent is a function where the last argument is a situation. Traditionally, fluents are predicate functions: $\texttt{box\_on\_table}(s)$, $\texttt{on\_table}(\texttt{box}, s)$, or $\texttt{on}(\texttt{table,box,}\, s)$. We extend the range to take on any real-value: $\texttt{num\_boxes\_on\_table}(s) \in \mathbb{R}$. A set of fluents is denoted $F = \{f^{(\alpha)}\}_{\alpha \in \{1,\ldots,k\}}$, whereas a fluent-vector is denoted by:

$$\vec{F}(s) = \begin{pmatrix} f^{(1)}(s) \\ f^{(2)}(s) \\ \vdots \\ f^{(k)}(s) \end{pmatrix} \tag{5.1}$$

Definition 14 (Internal Fluents and Internal Actions)**.** All fluents accessible to an agent may be thought of as internal, or cognitive, instead of physical. Correspondingly, actions that change internal fluents are called internal actions. For example, let $\sigma : \Omega_o \to \Omega_a$ be a function that extracts internal actions from observations. Then, σ is akin to "sensing" the environment, and updating an internal fluent. Therefore, the cognitive state (i.e. internal fluents) of an autonomous agent may diverge from physical reality.

Theorem 1 (Fluent-change)**.** Internal fluent change due to actions. *Proof*: An action a at a situation s leads to a new situation $s' = do(a, s)$. Since a fluent is a function on a situation, a changing situation affects the fluent.

Definition 15 (Action Pattern)**.** As shown in equation 5.2, we introduce a function A which we call an action pattern, that depends on both the situation and the next observation.

$$A(s, o; a) := \text{do}(a \circ \sigma(o), s) = s' \tag{5.2}$$

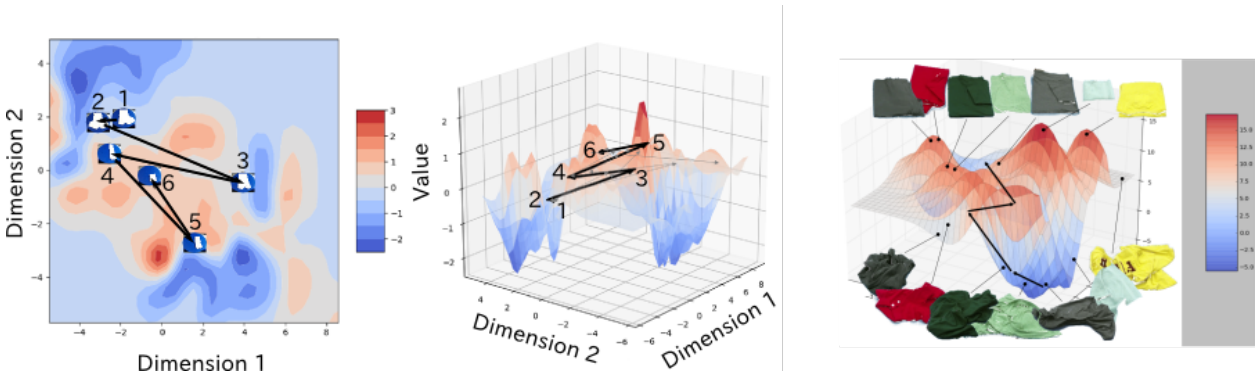


Figure 5.1: The utility landscape visualizes rankings of spatial configurations. Multiple cloth-folding demonstrations are parsed from videos, and each cloth is represented by a 14-dimensional vector of fluents. In this figure, we reduce the dimensionality for visualization using MDS, where the z-axis indicates the utility. After watching human demonstrations, the wrinkled states of a cloth end up with low utility, while well-folded clothes have high utility. The black arrow on the landscape traces the clothing situation during a single video demonstration.

Assume observations are not deterministic, but instead sampled from a probability distribution over the set of all observations Ω_o given a situation s. An action pattern generates atomic actions:

$$\{\vec{a} \mid \vec{a} \leftarrow A(s, o), o \sim P(\Omega_o|s)\} \tag{5.3}$$

We say a task is *resolved* by identifying a sequence of actions $\vec{a}$ that achieve the goal. The goal may not be unique or even well-defined, so we instead use the term *utility* to measure preference of being at a situation.

Definition 16 (Utility)**.** The utility $\hat{\mathcal{V}}$ is a function on the situation s, but for computational reasons, we assume the utility depends only on the relevant fluents $\vec{F}$ (see equation 5.4). Figure 5.1 shows a visualization of the utility landscape of clothes for a cloth-folding task.

$$\begin{aligned} \hat{\mathcal{V}} &: \Omega_s \to \mathbb{R} \\ \mathcal{V} &: \vec{F}(s) \mapsto v \in \mathbb{R} \end{aligned} \tag{5.4}$$

As discussed earlier, the model $\mathcal{M}$ of a task is characterized by a stochastic context-sensitive

grammar $\mathcal{G}$ of valid actions and the utility function $\mathcal{V}$.

$$\mathcal{M} = \langle \mathcal{G}, \mathcal{V} \rangle \tag{5.5}$$

Learning both the grammar and the utility are sufficient for deductive task-planning. We examine the likelihood of the model satisfying observations $\mathcal{O}$ in equation 5.6.

$$\begin{aligned} P(\mathcal{M}|\mathcal{O}) &= P(\mathcal{G}, \mathcal{V}|\mathcal{O}) \\ &= P(\mathcal{G}|\mathcal{O})\ P(\mathcal{V}|\mathcal{O}, \mathcal{G}) \end{aligned} \tag{5.6}$$

5.1.1 Model of utility $\mathcal{V}$

If situation s_a is preferred over s_b, we denote it by $s_a \succ s_b$. An ideal ranking function $\mathcal{R} : \Omega_s \to \mathbb{R}$ ranks situations, such that $s_a \succ s_b \ \Rightarrow\ \mathcal{R}(s_a) > \mathcal{R}(s_b)$. On the other hand, a utility function $\mathcal{V}$ introduces a slack variable $\xi > 0$:

$$\mathcal{V}(s_a) - \mathcal{V}(s_b) > 1 - \xi \tag{5.7}$$

Let $\mathcal{D} = \{f^{(1)}, f^{(2)}, ...\}$ be a dictionary of fluents, each with a latent potential function $\lambda : \mathbb{R} \to \mathbb{R}$. Using a sparse coding model, the utility of a situation is estimated by a small subset of relevent fluents $F = \{f^{(1)}, f^{(2)}, ..., f^{(k)}\} \subset \mathcal{D}$ and the corresponding set of potential functions $\Lambda = \{\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(k)}\}$.

$$\mathcal{V}(s; \Lambda, F) = \sum_{\alpha=1}^{k} \lambda^{(\alpha)}(f^{(\alpha)}(s)) \tag{5.8}$$

The fluents F are pursued one-by-one from $\mathcal{D}$ to minimize ranking violations ξ. Each potential function is approximated by a piece-wise-linear function with smoothness $\int_x \lambda''^{(\alpha)} dx$ taken into account [70, 71, 64].

5.1.2 Model of possible actions $\mathcal{G}$

Since actions are hierarchical in nature and long-term plans require foresight, we employ a stochastic context-sensitive grammar. Such generative models typically apply production rules defined in a graphical network. We outline below the graphical structure and semantics, based on the Attributed And-Or Graph (AOG) representation [8, 72].

Definition 17 (Attributed And-Or Graph)**.** A graph with access to fluents F (sometimes called *attributes*) and the following properties about its nodes (N) and edges (E).

- There are 3 types of nodes, denoted by sets $N = N_{AND} \sqcup N_{OR} \sqcup N_{TERMINAL}$.
- Each non-terminal node triggers a production rule.
  - The production rule for AND-nodes, ρ_{AND}, decomposes the node N into disjoint child nodes (the set of child nodes that descend from N is denoted $Ch(N)$):

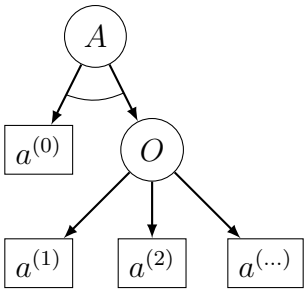
$$\rho_{AND} : N \to (Ch(N), Ch(N)) \tag{5.9}$$

  - The OR production rule decides a child given fluents:

$$\rho_{OR} : (N, \vec{F}) \to Ch(N) \tag{5.10}$$

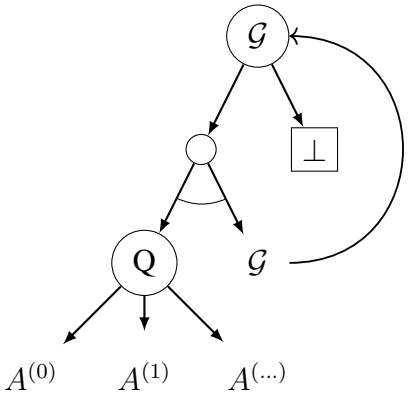
- The terminal nodes $n \in N_{TERMINAL}$ may modify the fluent attributes F.

An action pattern A models hierarchical compositions and variations of atomic actions. The actions $a \in \Omega_a$ are the terminal nodes of A. Shown below is a simple And-Or graph that may model $A = \{(a^{(0)}, a^{(1)}), (a^{(0)}, a^{(2)}), ...\}$. An AND-node is indicated by a curved line that bridges its children. A TERMINAL-node is indicated by a rectangle.



The graph $\mathcal{G}$ is constructed by n disjoint subgraphs $A_1, ..., A_n$. Each A is an action-pattern that represents a stochastic composition of atomic actions. Let $H \subset \mathcal{G}$ be the remaining graph structure, and let F be the set of fluents. Then, the following characterizes the graph.

$$\mathcal{G} = \langle A_1, A_2, ..., A_n, H, F \rangle \tag{5.11}$$



The production rules for OR-nodes depend on the fluents. For example, the node labelled Q is an OR-node, which selects an action pattern A, but this selection depends on the current assignment of fluents F. Performing an action sequence causes fluents to change, which in turn changes the behavior of Q.

Theorem 2 (Turing Complete)**.** In the supplementary material we provide an implementation of the Rule 110 automaton [73] to prove this mechanism is Turing Complete.

Definition 18 (Policy)**.** We define policy as a function $Q : s \mapsto A$ that maps a situation s to an action pattern A.

5.1.3 Planning

Influenced by previous work [74, 75] in decoupling utility from actions, this chapter proposes a control formulation to unify low-level mechanics with high-level preference. In deductive planning, the optimal selection of actions achieves a goal with minimum cost. We define the cost of a sequence of actions by the change in potential energy (i.e. utility of resulting situation) and kinetic energy (i.e. action cost). With that in mind, an agent must use their available actionable information to maximize utility,

An action a is instantiated by some concrete implementation u. For example, the dialogue action to say a "greeting" may be the utterance "hi there!". As another example, the robot action to move an arm from one point to another may be implemented by inverse kinematics. An agent performs a series of motions $u_{[1:t]}$ to maximize expected utility $\mathcal{V}$ while minimizing energy K, $u^*_{[1:t]} = \arg\max_{u_{[1:t]}} \big(\mathcal{V}(s_t) - \mathcal{V}(s_0) - K(u_{[1:t]})\big)$. The critical points may be found by taking the partial derivative of the utility. Using the chain rule, we factor the dynamics equation into 3 parts, shown below.

$$\frac{\partial \mathcal{V}}{\partial u_{[1:t]}} = \frac{\partial \mathcal{V}}{\partial F} \frac{\partial F}{\partial a_{[1:t]}} \frac{\partial a_{[1:t]}}{\partial u_{[1:t]}} \tag{5.12}$$

This allows us to study the utility-function $\mathcal{V}$ without considering the And-Or Graph $\mathcal{G}$ structure of how actions cause fluents to change.

5.2 Experiments & Results

We perform three sets of experiments to cover the various types of internal actions, and study each of them more concretely under the same task-oriented programming language.

1. **Mathematical Actions** - theorems driven by axioms. For example, incrementing an integer x is the action $x \leftarrow x + 1$. This increment action may repeat arbitrary number of times. With just a few valid mathematical operations, we can implement the greatest common divisor

(GCD) algorithm, as shown in Section 5.2.1.

2. **Empirical Actions** - patterns of reproducible measurements. For example, folding an article of clothing in half changes fluents. However, repeating that action may change the fluents in unpredictable ways, due to the nature of real-world interactions. We experiment with modelling a cloth-folding task on a robot in section 5.2.2; furthermore, we learn an interpretable task-model from few observations in section 5.2.2.

3. **Cognitive Actions** - mental mechanics. For example, remembering that a customer in a bakery wants a particular cake may be indicated by $flavor \leftarrow$ "chocolate". The rules for when or how actions may be set is very flexible. We show how to incorporate modelling cognition into our framework in section 5.2.3; and lastly, learn an interpretable chat-bot policy from few observations in section 5.2.3.

The model $\mathcal{M} = \langle \mathcal{G}, \mathcal{V} \rangle$ specifies the possible actions $\mathcal{G}$ and the utility $\mathcal{V}$, which are represented using a programming language whose grammar is specified in A [76]. The corresponding interpreter of this language is provided in the supplementary materials; therefore, we may interchange the words *model* and *code*. Then, *inference* is the process of running the code $\mathcal{M}$, and *learning* is the process of synthesizing code from few training examples. The following subsections walk through these three practical domains.

5.2.1 Mathematical Actions (Applied to Constraint Solving)

The greatest common divisor (GCD) of two numbers is the largest number that evenly divides both of them. For example, the GCD of 8 and 12 is 4. Algorithm 1 is a well-known implementation of GCD using traditional programming notions.

Although the Euclidean Algorithm is both succinct and efficient, it is not immediately intuitive. An algorithm trace reveals that there are only 2 instructions that modify variables: $a \leftarrow a - b$ (line 4) and $b \leftarrow b - a$ (line 6). Since we're interested in a way to implement GCD without needing to be an expert in the domain, we take note of all actions that may change variables, and list them out.

Algorithm 1 Euclidean Algorithm for Greatest Common Divisor

```
1: procedure GCD(a, b)
2:     while a ≠ b do
3:         if a > b then
4:             a ← a − b
5:         else
6:             b ← b − a
7:         end if
8:     end whilereturn a
9: end procedure
```

The set of relevant actions together with a utility function on situations defines the GCD procedure, shown in Algorithm 2, which is implemented using the syntax from A.

Running a solver on Algorithm 2 is not as computationally efficient; however, the programming language allows for easy modifications. For example, by changing the utility definition (lines 5-10), we can intuitively change the GCD procedure to instead solve for the least common multiple (LCM) of a and b, as shown in Algorithm 3.

By only changing the utility, we were able to reprogram the algorithm to perform a different task. This toy-example illustrates how we may begin to think about solving problems in other domains, such as robot cloth-folding or virtual chat-bot.

5.2.2 Empirical Actions (Applied to Robot Folding Clothes)

An artificial agent must first identify the goal of a task in order to evoke a series of actions to achieve it. Autonomous planning traditionally requires that a domain-expert manually define a heuristic-, fitness-, or loss-function to estimate progress towards the goal. As such, the utility of a situation ends up depending on hand-designed representations of the state space, such as propositional variables. We refer to such interpretable time-varying features as fluents.

In practice, the modeling of human utilities for achieving a task hinges critically upon the choice of relevant fluents. In fact, with appropriate fluents (i.e. with a suitable representation of the data and the state) planning becomes easy, almost trivial. With the wrong fluents, generalization of plans to novel situations becomes intractable, perhaps even impossible. If we represent video

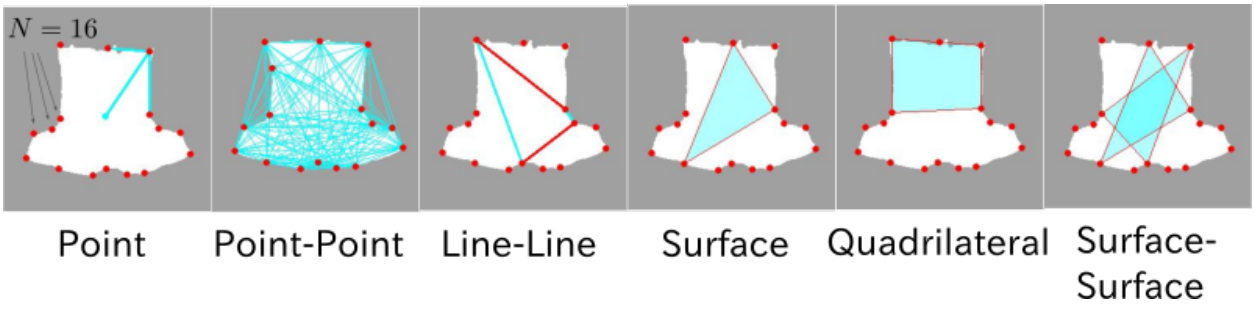


Figure 5.2: From tracking 16 keypoints on an article of clothing, there is no shortage of geometric relationships. Some may be less interpretable than others, but nothing is completely out of bounds of human interpretability.

demonstrations of a task through Fourier amplitude spectra or even autoencoder hidden states, the task might be learnable due to the comprehensive representation, but the correlation to human preferences is vague at best. In general, robust, accurate, and interpretable extraction of fluents is a crucial and difficult problem in modeling tasks.

We tackle the problem by first preprocessing our dataset in the following manner.

1. *Normalize perspective* - Geometric relationships are sensitive to the camera angle, so we transform the images such that the cloth appears to be seen from a top-down perspective. Given four points on the table, we calculate a transformation matrix to shift them to the four corners of the image. The same transform technique is applied across different images.

2. *Obtain binary mask* - Once the perspective has been warped, we begin to create a binary mask of the shirt. We start by segmenting the image and taking the largest contour to obtain a refined mask using the GrabCut algorithm [30].

3. *Extract 16 keypoints* - From the outer bounding box of the clothing, we identify points on the its contour by their four cardinal directions (e.g. North, South, East, West), the four intercardinal directions (e.g. North-East, etc.), and eight more divisions (e.g. North-North-East, etc.).

The keypoints enable many types of geometric relationships, as shown in figure 5.2. For in-

stance, the distance between the left-most and the right-most points may indicate the shirt's width, which is an interpretable fluent. There are thousands of fluents generated by the 16 keypoints (with 4 additional bounding box extrema points), and selecting the relevant ones is like finding a needle in a haystack. First, in section 5.2.2, we represent robot cloth-folding in our language, then in section 5.2.2 we simultaneously learn the utility function and relevant fluents.

Inference

We run the cloth-folding task experiments on a two-armed industrial robot. Each arm consists of 7 degrees of freedom that are manipulated through inverse kinematics relative to the robots frame of reference. Algorithm 4 outlines how to encode the task of cloth-folding in an interpretable way. The `utility` and `do` functions in the algorithm are defined externally. Results are summarized in figure **??**.

Learning

Given 45 videos of people folding clothes using different techniques, we model the utility function[77, 78]. The Ranking Pursuit algorithm, outlined in section 5.1.1, pursues from a medley of fluents one-by-one in a greedy fashion to minimize pair-wise ranking violations in the training dataset. Each fluent utility λ is estimated by a piecewise linear function. Once a handful of fluents are selected from this huge dictionary, we save the model parameters (Λ, F) for inference. Figure 5.3 shows our experiments in further detail.

We further evaluate our utility model against a similar framework called Inverse Reinforcement Learning (IRL). IRL is the process by which a reward function is recovered from demonstrations in a Markov Decision Process[67]. Specifically, we show the value-function in IRL is inferior to our ranking-based utility-function.

In order to prepare the data for use in the maximum entropy formulation of IRL[79], we first cluster the states (with K-Means). These cluster centroids discretize the possible folded states of the clothes. We construct a modified version of the gridworld MDP by normalizing this data, then

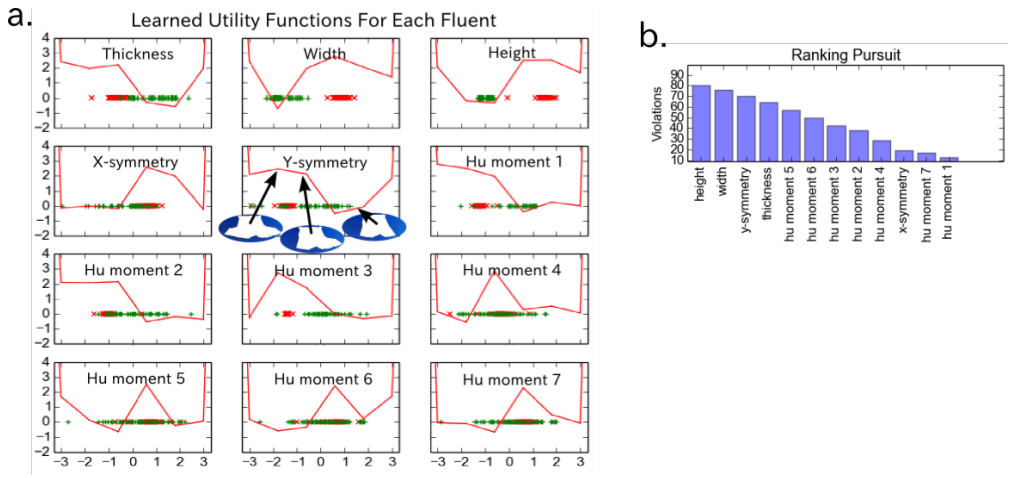


Figure 5.3: **(a)** The 12 curves represent the negative utility function ($-\lambda$) corresponding to each fluent. The functions are negated to draw parallels with the concept of potential energy. Red marks indicate initial fluent values, which the learned model appears to avoid, and the green marks indicate fluent values of the ending fluent values, which the learned model appears to favor. Notice how the y-symmetry potential energy decreases as the cloth becomes more and more symmetric. By tracing the change in utilities of each individual fluent, the robot can more clearly explain *why* it favors one state over another. **(b)** The ranking pursuit algorithm extracts fluents greedily to minimize ranking violations. As shown in the chart, the top 3 most important fluents for the task of cloth-folding are height, width, and y-symmetry, which are inferred from the geometric relationships.

pass the action trajectories into IRL to find the reward function. Once we have the reward function, we attempt to recover the value for each state space. The sum of the rewards up to and including the state is calculated by multiplying the reward of the state by a discount factor (0.9) to the power of the time step, averaged over the number of paths that traverse that state.

We ultimately must limit this to the observed trajectories, as it is computationally unfeasible to calculate every single path. For a 20x20 gridworld problem, the number of possible paths is lower bounded by $\binom{40}{20}$, or 137 billion. Given other possible actions, start states, and end states, the number of paths exponentially increases. While we can apply our knowledge of cloth-folding to restrict the state and action spaces, we cannot make general assumptions due to the nature of varying domains. The issue, then, with this method is that only those states and actions seen in the expert demonstrations are considered. This is in contrast to our ranking-based utility-model, which is capable of considering states and actions regardless of whether or not they appeared in our demonstrations.

Inverse Reinforcement Learning potentially regards intermediate states with higher reward values than the final state. This can occur anytime the same state is shared by multiple action trajectories. As a result, the reward function cannot be used by our fluent dynamics framework to suggest plans. As seen in Figure 5.4, this reward relationship is capable of throwing off the utility of the final state, such that the utility is improperly considered to be lower than a previous state. During task planning, our agent would then attempt to end at the previous state, never actually reaching the true goal state.

Although IRL's limitations in dimensionality have been mitigated by recent work in high-dimensional function approximation, the issue of needing a well-defined action set is a persistent weakness [80, 81]. The time complexity of function approximation in continuous high dimensional state spaces still scales linearly with the cardinality of the action set, showing the intertwined nature of the action set and IRL utility learning.

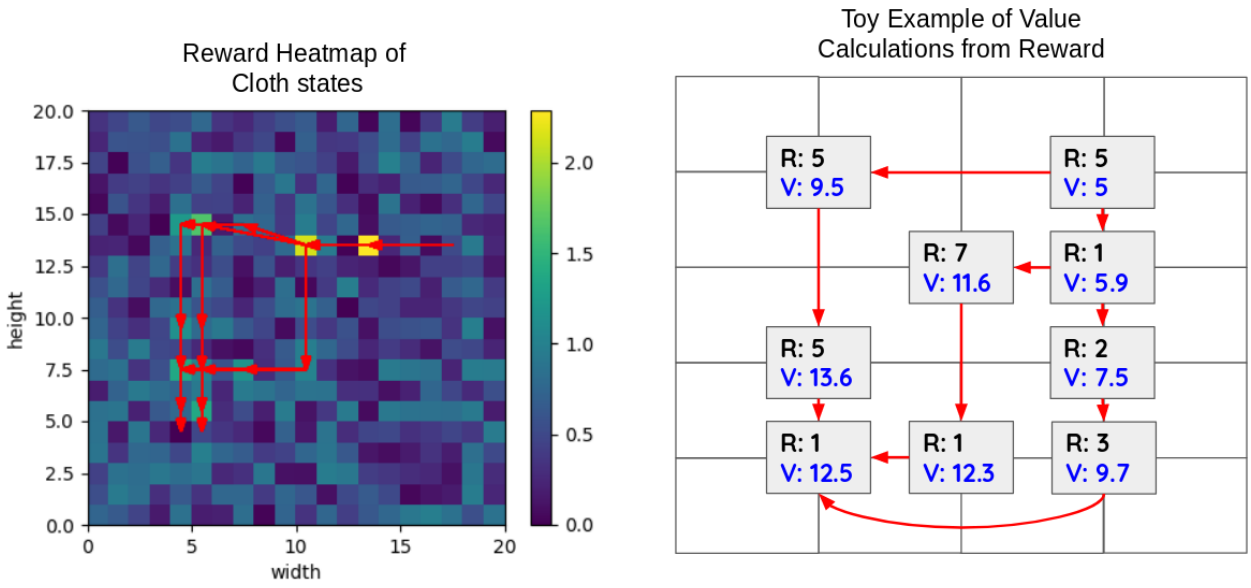


Figure 5.4: The learned reward R and value V show the limitations of using IRL for learning a utility over states for long-term planning scenarious such as cloth-folding. From the toy example on the right, we see that an intermediate step may be regarded higher value than the final step.

5.2.3 Cognitive Actions (Applied to Chat-bot Selling Items)

In this last set of experiments, we further emphasize the universality of the representation by applying the language to a chat-bot domain. In section 5.2.3, we first take a look at how a content-writer might author dialogue policies to craft interactive experiences. Then, in section 5.2.3, we show how to learn dialogue policies from a couple transcripts, so that dialogue policies may be generated automatically.

Inference

Based on work by Morbini et al. [82], we implement a goal-oriented chat-bot responsible for selling baked goods at a store. The dialogue policy, written in our language, is supplied in B. Our chat-bot behaves identically to its reference implementation. More interestingly, in the next section, we will synthesize the dialogue policy from few examples.

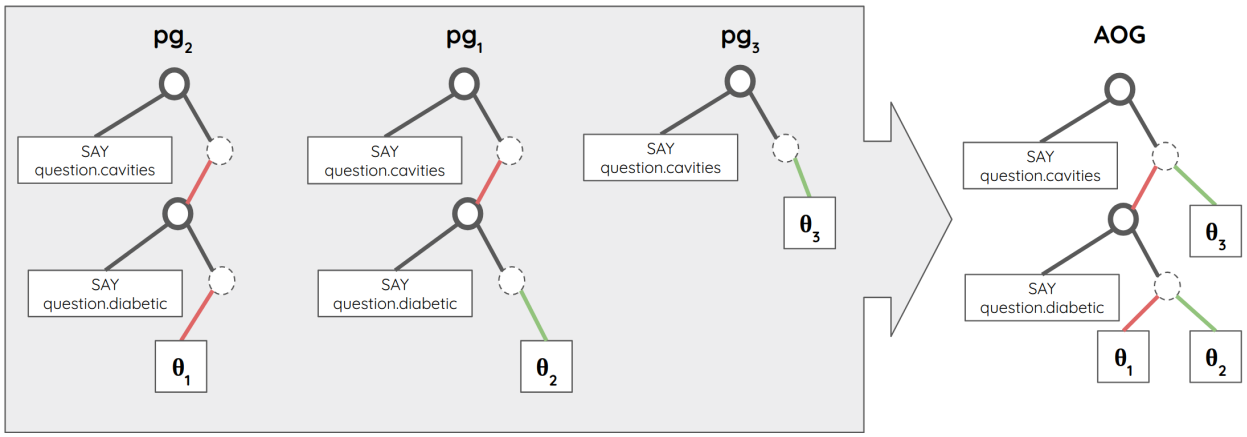


Figure 5.5: Each parse-graph (pg) is extracted from a dialogue transcript, and all-together merged into an AOG. OR-nodes, shown by a dotted circle, represent when the system is listening for a user's response. The color of the edge from an OR-node classifies the intent of the response, which in this case may be a positive or negative acknowledgement.

Learning

Given transcripts of dialogue, we learn the model $\mathcal{M}$. As shown in C, each utterance in a dialogue transcript has a corresponding intent (i.e. dialogue act), and a corresponding topic label. We construct a linear policy (i.e. parse-graph) per topic, where each user utterance corresponds to an OR-node selection, and a sequence of actions is strung together by an AND-node. Multiple parse-graphs in a topic are glued together naturally, as shown in figure 5.5.

The parameters $\Theta = \{\theta_1, \theta_2, ...\}$ characterize $\mathcal{M}$, where each θ corresponds to an atomic action a, such as $cavities \leftarrow True$. Assuming there are only 20 fluents, the number of Boolean actions is 40. Let $\mathcal{A}$ be the set of action patterns in $\mathcal{G}$, each containing no more than 3 atomic actions. Therefore the parameter-space is upper bounded by $|\Omega_\Theta| = 40^{3|\mathcal{A}|}$. We implement an algorithm we call Plan-MCMC to solve for Θ, outlined below.

1. Randomly initialize Θ

2. Iterate

   (a) Generate new candidates, $\{\Theta_1, \Theta_2, ..., \Theta_k\} \leftarrow P(\Theta'|\Theta)$. The transition from Θ to Θ'

is simply a modification of one action θ.

(b) For each Θ_i:

i. Sample parse-graphs, $\{\vec{a}_{syn}, ...\} \leftarrow \mathcal{M}_{\Theta_i}$

ii. Compute the histogram h of costs for each synthesized sample by comparing it to the observations, $cost(\vec{a}_{syn}) = \min_{\vec{a}_{obs}} Dist(\vec{a}_{obs}, \vec{a}_{syn})$. The distance is computed by best global pairwise alignment [83] (with the following scoring scheme: match 1, mismatch -2, gap -1).

iii. Normalize $\int_{c=0}^{B} h(c) = 1$, where B is the number of bins (e.g. $B = 20$).

iv. The acceptance probability of selecting Θ is proportional to $\int_{c=\epsilon}^{B} h(c)$, where $0 < \epsilon < B$ (e.g. $\epsilon = B/3$).

(c) Update Θ based on acceptance probability

In figure 5.6, we compare our approach against other techniques on both a training dataset of 4 examples and a test dataset of 3 examples. One such alternative technique is the N-Gram model (specifically $N = 4$), which builds frequency counts of all $1..N$ contingent pairs of intents, and then uses it to stochastically synthesize a new sequence of intents. Another common technique is to use a sequence-to-sequence (Seq2Seq) neural network on many example sequences, which typically fares well with big-data, but not so well on just 4 training examples. We also measure the performance of various human content-writers, indicating the optimal performance on training data. Lastly, we perform an experiment where our model hands off the learned policy to a human content-writer for making further edits, and this approach outperforms all other experiments on both the training and test dataset, showcasing the usefulness of an interpretable task model.

5.3 Discussion

Learning the utility of a task is essential for generalization. When fluents are chosen carefully, the utility function become interpretable. The central problem is then to find fluents of a task that

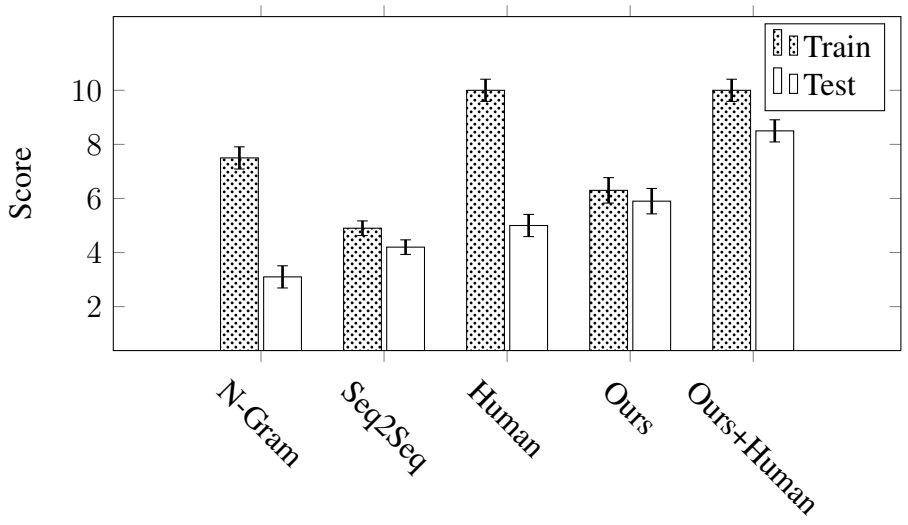


Figure 5.6: The N-Gram model memorizes the training data and ends up performing poorly against the test data. Asking a human to author the dialogue policy ends up with excellent performance on the training data. However, even a human has trouble with generalizing dialogue policies. Our model's generalizability performs comparably to a human. When a human context-writer improves on our generated code, we see the performance increase greatly.

are both relevant and interpretable. No doubt, interpretable representation of a task is one of the ultimate goals in the field of artificial intelligence, and we do not claim to have solved the problem. However, this chapter introduces the ranking pursuit algorithm that operates on a huge dictionary of interpretable fluents to identify relevant ones. Moreover, the Plan-MCMC algorithm introduces a way to synthesize the representation of a task given anonymous interpretable fluents.

With the complete fluent-dynamics framework expressed in equation 5.12, we aim to formalize the interdependence between utility, task, and motion. The utility drives the agent to perform sub-tasks (answering "why" an action is taken), whereas a sub-task instructs the agent "what" actions to take. Decoupling these two concepts allows us to learn the utility without an agent, or learn the available actions without accounting for the utilities. We show the power of mixing-and-matching utilities and actions in section 5.2.1, and in doing so, we utilize a new language defined in A.

5.4 Related Work

Learning a task or skill from visual demonstration is a widely applicable and thoroughly studied problem [53]. Task representation typically requires a definition of the goal state, which has been implemented in various ways. Below, we compare and contrast common approaches.

Modeling the prior: The prior probability expresses the belief based on previous knowledge. Traditionally, the knowledge of the goal was hand-designed by domain-experts [84]. Algorithms that automate the construction of priors from small training data are explored by [27], and were studied on natural images. The prior is typically modeled as a Gibbs distribution with learned potential functions. In some applications, kernel density estimates (KDE) are a common heuristic to model the prior [85].

However, estimating the prior based off matching frequencies is a form of inductive learning. In our work, we pursue relevant fluents that generalize beyond the scope of demonstrations.

Maximizing the likelihood: Bayesian formulations [42] assume a prior model of the goal (G), and use Bayes' Theorem to explain the relationship between the posterior and likelihood of the data ($\mathcal{X}$), $P(G \mid \mathcal{X}) = P(\mathcal{X} \mid G)P(G)/P(\mathcal{X})$. The maximum likelihood mindset has been used to justify imitation learning, where the agent finds model parameters to explain each human action [67].

For example, in most imitation learning frameworks, the learner assumes the system dynamics follow the assumptions of a Markov Decision Process (MDP) [75], and a reward function that maximizes the observed sequences of actions is sufficient to model the task. In this chapter, we learn to model a task without matching human action sequences.

Non-Markovian planning: The Markov assumption simplifies computation, but leads to troublesome limitations [51, 52]. On the other hand, Bayesian Program Learning (BPL) deals with learning how to connect compositional building blocks of a more complex target pattern to synthesize plans [86]. The BPL and AOG representations are both Non-Markovian generative models used for planning. Comparing representations between the two is beyond the scope of this chapter.

Matching human utilities: The utilitarian perspective infers the total order of a set of alternatives based on a training dataset of partial orders [64]. Unlike the local utility function in MDPs, which are defined by aggregating local rewards based on beliefs of the current state, the utilitarian ranking function (or global utility function) assigns a preference score to the state independent of the agent's set of actions.

In particular, the closest related work includes a study in modeling human preferences over internal properties of physical tools from video demonstrations [48]. Our work differs in three ways: 1) relevant fluents are pursued by simultaneously minimizing reconstruction error and ranking violations; 2) we learn from external fluents, such as states of the world, instead of internal fluents, such as states of the agent; 3) the utility function drives robot plans.

There has also been previous work on recommender systems designed to match human preference over pairs of clothing images that complement each other in style [87].

Algorithm 2 Utility-Based Algorithm for Greatest Common Divisor

Initialize ▷ Set up the fluents a, b, and c, with the semantic meaning $c = \gcd(a, b)$
    $a_0 \leftarrow 8$
    $b_0 \leftarrow 12$
    $c_0 \leftarrow$ **null**
end Initialize

▷ Define the utility function
Utility ($c \mid a_0$) ▷ If c divides a_0, accumulate 1 reward
    reward $+1$
end Utility
Utility ($c \mid b_0$) ▷ If c divides b_0, accumulate 1 reward
    reward $+1$
end Utility
Utility ($c \mid a_0$ **and** $c \mid b_0$) ▷ If c divides both, accumulate c reward
    reward $+c$
end Utility

▷ Allow the following actions
Action
    $a \leftarrow a + b$
end Action
Action
    $b \leftarrow b + a$
end Action
Action
    $a \leftarrow a - b$
end Action
Action
    $b \leftarrow b - a$
end Action
Action ▷ Return the result
    $c \leftarrow a$
    $\perp$
end Action

Algorithm 3 Utility Definition for Least Common Multiple

Utility ($a_0 \mid c$) ▷ Favor situations when a_0 divides c
    reward +1
end Utility
Utility ($b_0 \mid c$) ▷ Favor situations when b_0 divides c
    reward +1
end Utility
Utility ($a_0 \mid c$ **and** $b_0 \mid c$) ▷ Favor situations when both divide c
    reward $+1/c$
end Utility

Algorithm 4 Utility-Based Algorithm for Robot Folding Clothes

Initialize
    $keypoints \leftarrow$ **null**
end Initialize
▷ Compute utility using an external function
Utility
    reward +utility($keypoints$)
end Utility
▷ Allow the following actions
Action ($keypoints^* \neq$ **null**) ▷ Use sensors to set keypoints when available
    $keypoints \leftarrow keypoints^*$
end Action
Action
    $keypoints \leftarrow \text{do}(fold_{(v,\frac{1}{2})}, keypoints)$ ▷ Simulate a vertical fold in half
end Action
Action
    $keypoints \leftarrow \text{do}(fold_{(v,\frac{1}{3})}, keypoints)$ ▷ Simulate a different vertical fold
end Action
Action
    $keypoints \leftarrow \text{do}(fold_{(h,\frac{1}{2})}, keypoints)$ ▷ Simulate a horizontal fold in half
end Action
Action
    ... ▷ Remaining actions omitted for brevity
end Action

CHAPTER 6

Task-Oriented Programming Language

Complex interactive experiences beyond flowchart-like logic are difficult to author, and an agreed upon representation is still missing. However, there are multiple programming languages that have been designed for representing tasks. Among them, we brand our task-oriented language Dialogue Manager Programming Language (DMPL) which is designed specifically with conversational interfaces in mind.

One of the first languages, Planning Domain Definition Language (PDDL) [88] represents tasks by two files: a domain file, which lists available actions, and a corresponding problem file, which defines the initial fluents and goal specifications. Each action defined in the domain file has an entry-condition and a deterministic effect. Deterministic planning is a major limitation, since we're modeling tasks in interactive or unpredictable environments.

Probabilistic PDDL (PPDDL) [89] addresses that limitation by introducing stochastic effects, so that planning may account for probabilistic outcomes of each action. The PPDDL inference engine can solve for an action sequence that optimizes a user-defined utility function. Through some careful organization and design of the domain file, it is also possible, but not intuitive, to support hierarchical planning within PPDDL. On the other hand, DMPL is designed with hierarchical planning from the get-go, by pushing fluents to a stack once entering a new subroutine.

Unlike PPDDL, WebPPL and Stan [90] are general purpose probabilistic programming languages that offer a rich selection of statistical models and efficient MCMC sampling. These languages are used to specify general statistical models, which can model an autonomous agent in a stochastic environment. Since they're general purpose, like DMPL, WebPPL and Stan support user-defined functions and external interfaces to user-defined business logic.

Languages designed specifically for decision problems include IBAL [91] and Markov logic decision networks (MLDN) [92], which come with approximate inference solvers. DTProbLog [93] introduces exact solvers to compute the optimal strategy in a decision-theoretic probabilistic language, using Algebraic Decision Diagrams (ADD) [94]. However, DTProbLog is less concerned with sequential decision problems. These languages leverage the Markov Decision Process (MDP) framework, which requires a definition of immediate reward of transitioning from one state to another. Strategically, DMPL disallows users from defining MDP-like rewards and limits the utility function to be defined indirectly through listing pairs of preferred situations. Doing so alleviates the guess-work from the user, and reduces buggy code (most common being infinite-loops due to ill-defined rewards).

Missing from most probabilistic programming languages is the runtime of the agent. In other words, once an action-sequence is resolved, instructions for following through with the chosen action on an agent is run elsewhere, usually defined in a separate codebase. DMPL borrows ideas from DTGolog [95], where both the planner (simulation engine) and evaluator (runtime) consult the same user-written AOG (i.e. code). The AOG is used both to plan by sampling possible actions and to execute by traversing the same graph.

The abstract syntax tree of our task-oriented programming language is explicitly defined in this chapter. It follows the syntax of JavaScript Object Notation (JSON), so that it may easily be parsed by web technologies, for easy prototyping. Reusing the same JSON for planning and execution allows this language to support goal-switching, meaning the definition of the goal may be adjusted by the agent itself. Goals are not defined explicitly by a mathematical function (akin to PPDDL, etc.), but instead inferred (by RankSVM) from pair-wise preferences of situations. Moreover, since our language is designed for conversational interfaces, proper handling of interruptions (handing timeouts, barge-in, and global events) is an essential built-in component of the language.

As shown in the previous chapter, this language unifies hierarchical conformant planning (cloth-folding), mathematical optimizations (solving greatest common divisor), and hierarchical contingent planning (chat-bots). By reusing the same representation for both planning and execu-

tion, the programmer can use DMPL as a one-stop solution his or her domain.

6.1 Dialogue Manager Programming Language

This specification defines the syntax and semantics of a dialogue manager programming language (DMPL). The user specifies the states (i.e. fluents), how they change (i.e. actions), and which are preferred (i.e. utility). These three components characterize a task, so we call DMPL a task-oriented programming language. The interpreter of the language should resolve the task by identifying and executing actions that maximize utility.

This specification aims to decouple the representation of dialogue from its execution. For example, conversational interfaces employ artificial intelligence in different ways: finite state machines, path planning, or reinforcement learning systems. Regardless of the execution framework, the logical representation may be shared. An agreed upon representation of dialogue allows content writers to author and share conversational experiences without being distracted by the underlying runtime.

This document is limited to specifying the syntax and semantics for a task-oriented language for interactive behavior. One example application of DMPL is an authoring tool to be used on a regular basis by a non-technical content writer. Another use-case of DMPL is exporting and importing dialogue content authored by writers from different web platforms. Technologies closely related to these use-cases are in scope.

6.2 Language Syntax

6.2.1 Atomic Types

A literal is a terminal atom that may be assigned to a variable, used in a dictionary, or used as an operand to build an expression.

```
literal ::= string | number | boolean | 'null'
```

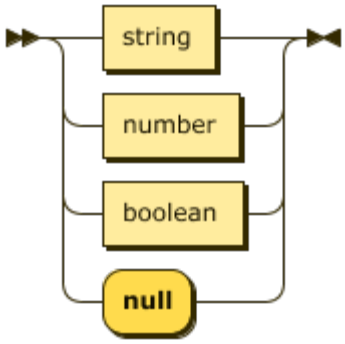


Figure 6.1: Literal

A boolean may take on values true or false.

```
boolean ::= 'true' | 'false'
```

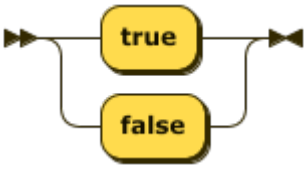


Figure 6.2: Boolean

A name is a sequence of characters conforming to those in the XML standard.

```
name ::= [ http://www.w3.org/TR/xml-names/#NT-NCName ]
```

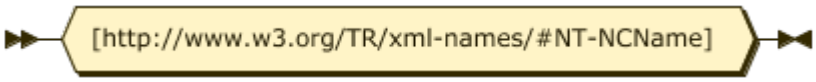


Figure 6.3: Name

A `quote_start` is the starting quotation mark, which indicates the beginning of a string.

```
quote_start ::= '"' | '`'
```

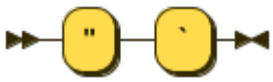


Figure 6.4: Start of string

A `quote_close` is the ending quotation mark, which indicates the termination of a string.

```
quote_close ::= '`' '"'
```

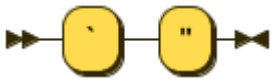


Figure 6.5: End of string

A string literal represents a finite sequence of characters. It is surrounded by starting and ending quotation marks.

```
string ::= quote_start name quote_close
```

Figure 6.6: String

A digit represents a real valued integer in the range [0-9].

```
digit ::= [0-9]
```

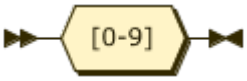


Figure 6.7: Digit

A number literal may represent either a positive or negative decimal value.

```
number ::= '-'? ( digit+ ( '.' digit* )? | '.' digit+ )
```

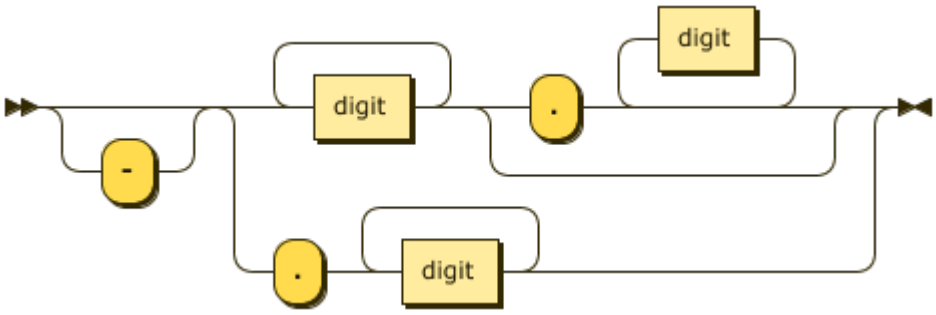


Figure 6.8: Number

6.2.2 Container Structures

A variable is a named reference to the result of evaluating an expression.

```
variable :: = '"' name '"'
```

Figure 6.9: Variable

Example 3. `"num_correct_answers"`

A variable should not be confused with a string, which uses backticks (‘) like so:

Example 4. `"`num_correct_answers`"`

A dictionary is a structure that maps data in key-value pairs. The keys and values are all evaluated. The keys can be either strings or variables that evaluate to strings.

```
dictionary : : = '{' ( string | variable ) ':' expression
                 ( ',' ( string | variable ) ':' expression )* '}'
```

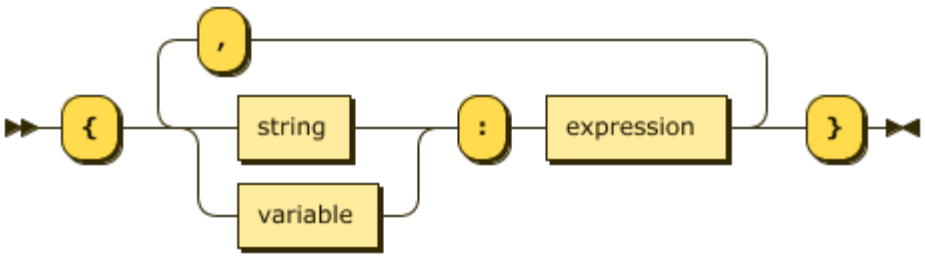


Figure 6.10: Dictionary

Example 5. `{"`age`": "x", "`score`": 10, "`name`": "`Joe`"}`

An expression may be evaluated, and that result is stored in memory to be referenced later. An expression may be denoted by a JSON array, in which case it is also called a function. The first element of the array represents the operator of the function. The operator can be either a string or a variable that evaluates to a string. All remaining elements of the array constitute the operands.

```
expression ::= literal
             | variable
             | dictionary
             | '[' ( literal | variable ) ( ',' expression )* ']'
```

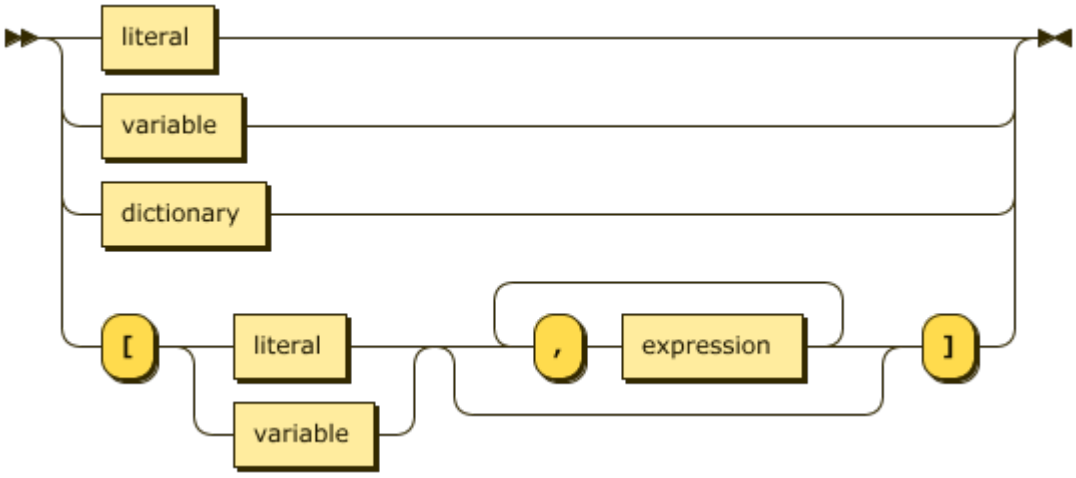


Figure 6.11: Expression

Table 6.1: Subset of DMPL expressions

Name	Return type	Operator	Example
Concat	string	`"++"`	`["++", "`hello `", "`world`"]`
Add	number	`"+"`	`["+", 1, 2]`
Subtract	number	`"-"`	`["-", 5, 3]`
Multiply	number	`"*"`	`["*", 2, 4]`
Divide	number	`"/"`	`["/", 1, 2]`
Modulo	number	`"%"`	`["%", 168, 2]`
List	list	`""`	`["", 1, 2, 3, 4, 5]`
Range	list	`"enumFromTo"`	`["enumFromTo", 1, 5]`
Equals	boolean	`"=="`	`["==", 1, 2]`
Not Equals	boolean	`"!="`	`["!=", 1, 2]`
Greater Than	boolean	`">"`	`[">", 2, 3]`
GTE	boolean	`">="`	`[">=", 2, 3]`
Less Than	boolean	`"<"`	`["<", 2, 3]`
LTE	boolean	`"<="`	`["<=", 2, 3]`
Logical And	boolean	`"&&"`	`["&&", true, false]`
Logical Or	boolean	`"||"`	`["||", true, false]`
Logical Not	boolean	`"!"`	`["!", false]`
Contains	boolean	`"in"`	`["in", 3, ["", 1, 2, 3]]`
Input check	boolean	`"input"`	`["input", "yes"]`
Return check	boolean	`"return"`	`["return"]`
Exists	boolean	`"exists"`	`["exists", "`is_greeted`"]`
Get By Index	dynamic	`"get"`	`["get", 0, ["", 1, 2, 3]]`
Pick	dynamic	`"pick"`	`["pick", ["", 1, 2, 3]]`

6.2.3 Statements

A statement may be a non-terminal such as do or fork, or a terminal such as effect. Optionally, statements may contain condition, await, or once flags.

```
statement ::= '{' ( condition ',' )?
              ( await ',' )?
              ( once ',' )?
              ( effect | do | fork ) '}'
```

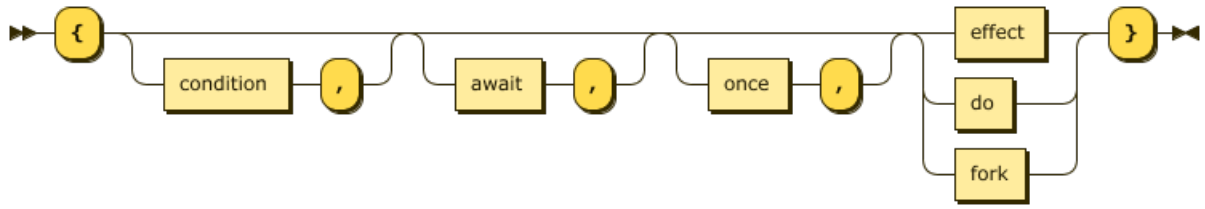


Figure 6.12: Statement

A condition is a boolean expression that runs the statement if the expression evaluates to true. A condition check takes priority over other parts of the statement. If a statement does not explicitly contain a condition, then that statement's condition is trivially true.

```
condition : : = '"if"' ':'  expression
```

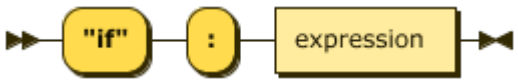


Figure 6.13: Condition

Example 6. `"if": [">", "num_wrong_answers", 3]`

An await blocks execution until a boolean expression evaluates to true.

```
await ::= '"await"' ':' expression
```

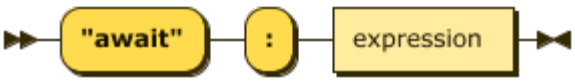


Figure 6.14: Await

Example 7. `"await": ["input"]`

A once flag specified whether the statement can only be run once. By default, the once flag is set to false.

```
once ::= '"once"' ':' boolean
```

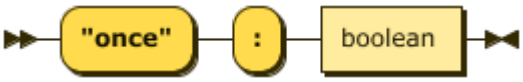


Figure 6.15: Once

Example 8. `"once": true`

An effect is a terminal node, meaning it does not produce more statements. There are 6 types of effects.

```
effect ::= act | set | def | run | use | pop
```

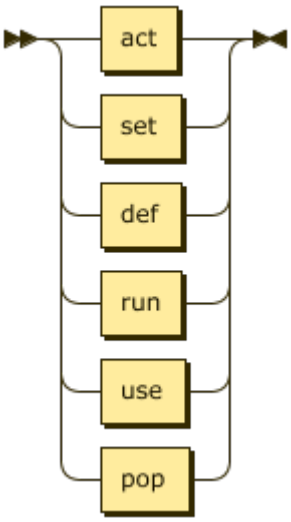


Figure 6.16: Effect

An act represents an action specified by evaluating the expression. The result is interpreted by a client. For example, the payload of the act statement may be represented using Behavior Markup Language (BML). The client is then responsible for realizing the behavior defined in the act message.

```
act ::= '"@act"' ':' expression
```

Figure 6.17: Act

Example 9.

```
"@act": {
  "`object`": "`tutor`",
  "`action`": "`say`",
  "`params`": {"`intent`": "`greeting`"}
}
```

A set updates the values of the variables, where the names of the variables and the updated values are specified by evaluating the corresponding expressions. The expression specifying the names of the variables is allowed to contain variables (although static code analysis and possible optimizations would be more difficult), and must evaluate to a string, a list consisting of only strings, or a dictionary consisting of only strings.

```
set ::= '"@set"'  ':' expression ',' '"val"' ':' expression
```

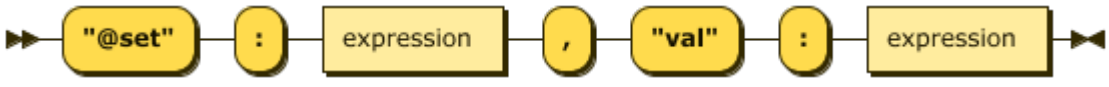


Figure 6.18: Set

Example 10. `"@set": "`is_user_greeted`", "val": true`

Example 11. `"@set": ["", "`is_user_greeted`", "`num_questions`"], "val": ["", true, 7]`

A def defines a new expression operator that can be used later in the code. The signature of the new expression operator is specified by an expression that must evaluate to a list of strings, which maps to the operator name followed by the argument names. These argument names are valid only for this scope, and may shadow global variables. The result is returned by a pop statement.

```
def ::= '"@def"' ':' expression ',' '"val"' ':' statement
```

Figure 6.19: Def

Example 12. `"@def": ["", "`inc`", "`x`"], "val": {"@pop": ["+", 1, "x"]}`

A run calls DMPL code in another component, optionally passing in arguments. The name of the called component is specified by an expression that must evaluate to a string. The current variables stored in memory are remembered later.

```
run ::= '"@run"' ':' expression ( ',' '"args"' ':' expression )?
```

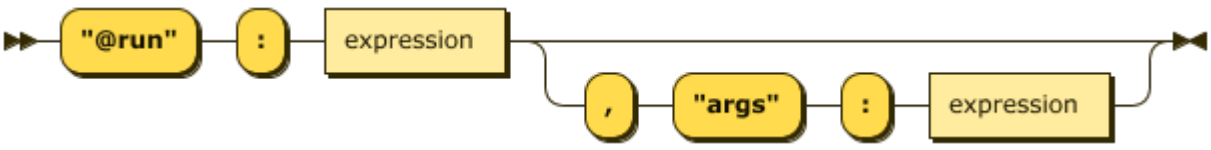


Figure 6.20: Run

Example 13. `"@run": "`Outro`"`

Example 14. `"@run": "`Question`",`
`"args": ["", "`What's the largest planet?`", "`Jupiter`

A use imports variables and expression operators defined in another component. The name of the imported component is specified by an expression that must evaluate to a string. The variable and expression operator names can be optionally specified, or all variables and expression operators will be imported. It's similar to from os import path notation in Python.

```
use ::= '"@use"' ':' expression ( ',' '"import"' ':' expression )?
```

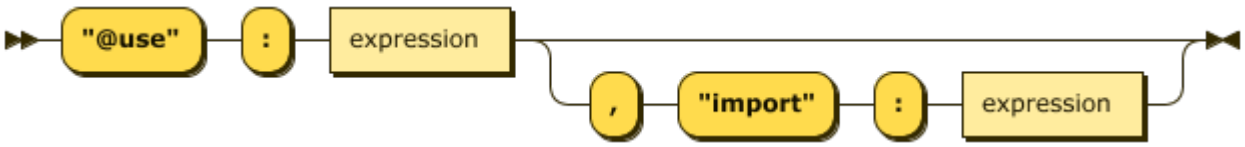


Figure 6.21: Use

Example 15. `"@use": "`MathExpressions`",`
`"import": ["", "`inc`", "`square`", "`exp`"]`

A pop quits execution on the current DMPL code, pops the fluent-state from the stack, and resumes execution from the previous run location.

```
pop ::= '"@pop"' ':' expression
```

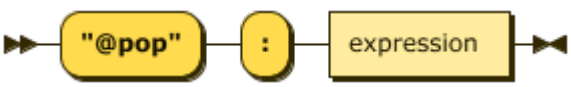


Figure 6.22: Pop

A do statement specifies a list of statements to be executed.

```
do ::= '"@do"' ':' '[' ( statement ( ',' statement )* )? ']'
```

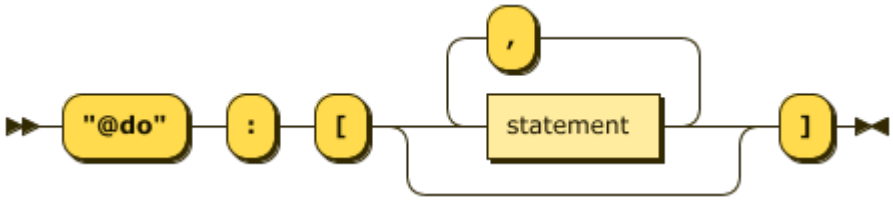


Figure 6.23: Do

Example 16.

```
"@do": [
  {"@act": "a"},
  {"@act": "b"},
  {"@act": "c"}
]
```

A fork statement specifies the branch-like behavior of the list of statements that follow. Only one is chosen, and the strategy to pick one may be provided using the scheme attribute. By default, a greedy scheme is used, meaning the first statement whose condition is met will be chosen. More complicated schemes, such as depth-first-search or Monte Carlo Tree Search may be indicated, leaving the implementation up to the interpreter.

```
fork ::= '"@fork"' ':' '['
    ( statement ( ',' statement )* )?
  ']' ( ',' scheme )?
```

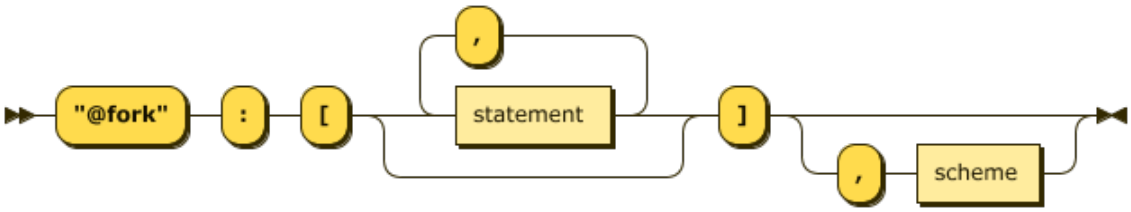


Figure 6.24: Fork

When no scheme is provided, a fork is essentially the traditional if/elif/else logical flow:

Example 17.

```
"@fork": [
  {"if": "is_user_greeted", "@act": "a"},
  {"if": [">", "num_wrong_answers", 3], "@act": "b"},
  {"@act": "c"}
]
```

Providing a scheme allows stochastic behavior to take place:

Example 18.

```
"scheme": {"depth": 3}, "@fork": [
  {"@set": "`do_action_1`", "val": "true"},
  {"@set": "`do_action_2`", "val": "true"},
  {"@set": "`do_action_3`", "val": "true"},
  {"if": "is_entry_condition_met",
    "@set": "`do_action_4`", "val": "true"},
  {"if": false, "@act": "`never reached`"}
]
```

A scheme is associated with a fork statement, and it describes the fork branch resolution strategy.

```
scheme ::= '"scheme"' ':' expression
```

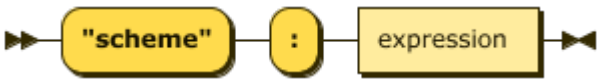


Figure 6.25: Scheme

Example 19. `{"depth": 3}`

6.3 Semantics

The runtime of this language operates on the JSON, starting from the root statement. It traverses down, performing actions corresponding to the terminal nodes, until there is a fork. Then, forks are resolved by their indicated scheme, and this process continues.

Once a path through the JSON is complete, the runtime starts over from the beginning. Also, if no entry-condition is met, then execution restarts from the root. It's safe to assume the code written by a user is embedded within a while-true loop.

The runtime is responsible for handling user input and system output in an asynchronous way. Every user input triggers a re-plan, allowing for interruptions to be handelled. System outputs, such as payloads of act statements, are sent to attached processes, as needed.

Appendices

APPENDIX A

Programming Language Syntax

A simplified syntax is specified in extended Backus-Naur form (EBNF) below; however, the complete language used in our experiments follow the more verbose syntax of the W3C report [76].

```
bool ::= "true" | "false"
digits ::= [0-9]+
num ::= ("." digits) | (digits "." [0-9]*)
str ::= [http://www.w3.org/TR/xml-names/#NT-NCName]
op_bool ::= "and" | "or"
op_num ::= "+" | "-" | "*" | "/"
op_num_check ::= ">" | "<" | "<=" | ">=" | "==" | "!=" | "|"
str_expr ::= str | (str_expr "+" str_expr)
num_expr ::= num | num_expr op_num num_expr
bool_expr ::= bool
            | ( bool_expr op_bool | "NOT" ) bool_expr
            | num_expr cmp num_expr
expr ::= str_expr | op_num_check | bool_expr
say ::= "say" str_expr
set ::= str_expr "<-" expr
or_node ::= "sense" str | ( "if" bool_expr "then" and_node )+
terminal_node ::= say | set
and_node ::= (terminal_node | or_node )+
action ::= "Action" ( "(" bool_expr ")" )? ( and_node )*
```

```
utility ::= "Utility" ( "(" bool_expr ")" )? "reward" num_expr
initialize ::= "Initialize" (set)*
program ::= initialize (utility)* (action)*
```

APPENDIX B

Example Implementation of a Chat-Bot

Algorithm 5 Utility-Based Chat-bot (Cake Vendor)

Initialize ▷ Initialize the fluents
    greeted $\leftarrow false$
    known_cake_type $\leftarrow false$
    known_sugar_sugarless $\leftarrow false$
    known_cake_flavor $\leftarrow false$
end Initialize
▷ The program tries to maximize the following utilities
Utility (greeted) ▷ The customer is greeted
    reward 10
end Utility
Utility (known_cake_type)
    reward 10
end Utility
Utility (known_sugar_sugarless)
    reward 10
end Utility
Utility (known_cake_flavor)
    reward 10
end Utility
▷ Only the following actions are available
Action (not greeted)
    say greeting
    greeted $\leftarrow true$
end Action
Action (greeted)
    say question.cake-type
    $sense$ cake-type
    known_cake_type $\leftarrow true$
end Action
Action (greeted **and** known_cake_type)
    say question.cake-flavors
    $sense$ flavor
    known_cake_flavor $\leftarrow true$
end Action
Action (greeted)
    say question.chocolate-cake
    $sense$ chocolate-cake
    known_cake_type $\leftarrow true$
end Action

APPENDIX C

Example Dialogue Transcript

Agent	Intent	Utterance
system	greeting.hello	Hello
system	greeting.welcome	Welcome to the store.
system	question.what.user-wants	What would you like?
user	**statement.cake**	**cake**
system	acknowledgement	ok
system	question.cavities	Do you have cavities?
user	**answer.no**	**no**
system	question.diabetic	Are you diabetic?
user	**answer.yes**	**yes**
system	statement.cake.sugarless	For you we will make only cakes sweetened with Xylitol
system	question.confirmation	Is that ok?
user	**answer.yes**	**yup**
system	question.cake.type	What type of cake would you like?
user	**statement.flavor.lemon**	**I want lemon**
system	apology.flavor	I'm sorry, that flavor is not available for the cake you requested.
system	answer.cake.flavor.sponge	We can make chocolate or amaretto sponge cakes
system	question.cake.flavor	What flavor would you like?
user	**statement.flavor.chocolate**	**chocolate, please**
system	statement.cake.ready.chocolate.sponge	Your chocolate sponge cake is ready!
system	greeting.bye	Bye

References

[1] K. Tu, M. Meng, M. W. Lee, T. E. Choe, and S.-C. Zhu, "Joint video and text parsing for understanding events and answering queries," *IEEE MultiMedia*, vol. 21, no. 2, pp. 42–70, 2014.

[2] Z. Si, M. Pei, B. Yao, and S.-C. Zhu, "Unsupervised learning of event and-or grammar and semantics from video," in *2011 International Conference on Computer Vision*, IEEE, 2011, pp. 41–48.

[3] A. Gupta, P. Srinivasan, J. Shi, and L. S. Davis, "Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 2012–2019.

[4] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 249–267, 2012.

[5] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[6] Y. Yang, Y. Li, C. Fermuller, and Y. Aloimonos, "Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] J.-W. Ha, K.-M. Kim, and B.-T. Zhang, "Automated construction of visual-linguistic knowledge via concept learning from cartoon videos," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[8] S.-C. Zhu and D. Mumford, "A stochastic grammar of images," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 4, pp. 259–362, 2006.

[9] P. Wolff, "Representing causation.," *Journal of experimental psychology: General*, vol. 136, no. 1, p. 82, 2007.

[10] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 2397–2403.

[11] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *The International Journal of Robotics Research*, p. 0 278 364 911 426 178, 2011.

[12] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 2, pp. 286–298, 2007.

[13] C. Liu, J. Y. Chai, N. Shukla, and S.-C. Zhu, "Task learning through visual demonstration and situated dialogue," in *AAAI'16 Workshop on Symbiotic Cognitive Systems*, 2016.

[14] N. Shukla, C. Xiong, and S.-C. Zhu, "A unified framework for human-robot knowledge transfer," in *AAAI'15 Fall Symposium on AI for Human-Robot Interaction (AI-HRI 2015)*, 2015.

[15] Y. Yamakawa, A. Namiki, and M. Ishikawa, "Motion planning for dynamic folding of a cloth with two high-speed robot hands and two high-speed sliders.," in *ICRA*, IEEE, 2011, pp. 5486–5491.

[16] R. P. N. Rao, A. P. Shon, and A. N. Meltzoff, "A bayesian model of imitation in infants and robots," in *In Imitation and Social Learning in Robots, Humans, and Animals*, Cambridge University Press, 2004, pp. 217–247.

[17] P. C. Wang, S. Miller, M. Fritz, T. Darrell, and P. Abbeel, "Perception for the manipulation of socks.," in *IROS*, 2011, pp. 4877–4884.

[18] A. Doumanoglou, A. Kargakos, T. Kim, and S. Malassiotis, "Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 2014, pp. 987–993.

[19] E. T. Mueller, in *Commonsense Reasoning*, E. T. Mueller, Ed., Morgan Kaufmann, 2006.

[20] G. W. Strong and Z. W. Pylyshyn, "Computation and cognition: Toward a foundation for cognitive science. cambridge, massachusetts: The mit press, 1984, 320 pp.," *Behavioral Science*, vol. 31, no. 4, pp. 286–289, 1986.

[21] H. Chen, Z. J. Xu, Z. Q. Liu, and S. C. Zhu, "Composite templates for cloth modeling and sketching," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, 2006, pp. 943–950.

[22] K. Tu, M. Pavlovskaia, and S.-C. Zhu, "Unsupervised structure learning of stochastic and-or grammars," in *Advances in Neural Information Processing Systems 26*, Curran Associates, Inc., 2013, pp. 1322–1330.

[23] A. S. Fire and S. Zhu, "Learning perceptual causality from video," in *Learning Rich Representations from Low-Level Sensors, Papers from the 2013 AAAI Workshop, Bellevue, Washington, USA, July 15, 2013*, 2013.

[24] J. Rekers and A. Schrr, "A parsing algorithm for context-sensitive graph grammars," Tech. Rep., 1995.

[25] C. Chao, M. Cakmak, and A. Thomaz, "Towards grounding concepts for transfer in goal learning from demonstration," in *Development and Learning (ICDL), 2011 IEEE International Conference on*, vol. 2, 2011, pp. 1–6.

[26] K. Zampogiannis, Y. Yang, C. Fermller, and Y. Aloimonos, "Learning the spatial semantics of manipulation actions through preposition grounding.," in *ICRA*, IEEE, 2015, pp. 1389–1396.

[27] S. C. Zhu, Y. N. Wu, and D. Mumford, "Minimax entropy principle and its application to texture modeling.," *Neural Computation*, vol. 9, no. 8, pp. 1627–1660, 1997.

[28] C. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 2, 1997, 1706–1712 vol.2.

[29] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, Sep. 2004.

[30] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut -interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics (SIGGRAPH)*, 2004.

[31] Neo4j, *Neo4j - the worlds leading graph database*, 2012.

[32] S. Chernova and A. L. Thomaz, "Robot learning from human teachers," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[33] L. She, S. Yang, Y. Cheng, Y. Jia, J. Y. Chai, and N. Xi, "Back to the blocks world: Learning new actions through situated human-robot dialogue," in *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014, p. 89.

[34] C. Xiong, N. Shukla, W. Xiong, and S. Zhu, "Robot learning with a spatial, temporal, and causal and-or graph," in *Proceedings of International Conference on on Robotics and Automation (ICRA 2016)*, Stockholm, Sweden, 2016.

[35] H. H. Clark, *Using language*. Cambridge university press, 1996.

[36] M. Pei, Z. Si, B. Z. Yao, and S.-C. Zhu, "Learning and parsing video events with goal and intent prediction," *Computer Vision and Image Understanding*, vol. 117, no. 10, pp. 1369–1383, 2013.

[37] J. Lave and E. Wenger, *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[38] J. Herrington and R. Oliver, "Critical characteristics of situated learning: Implications for the instructional design of multimedia," 1995.

[39] H. H. Clark and E. F. Schaefer, "Contributing to discourse," *Cognitive science*, vol. 13, no. 2, pp. 259–294, 1989.

[40] O. Lemon, A. Gruenstein, and S. Peters, "Collaborative activities and multi-tasking in dialogue systems: Towards natural dialogue with robots," *TAL. Traitement automatique des langues*, vol. 43, no. 2, pp. 131–154, 2002.

[41] B. J. Grosz and C. L. Sidner, "Attention, intentions, and the structure of discourse," *Computational linguistics*, vol. 12, no. 3, pp. 175–204, 1986.

[42] V. N. Vapnik and V. Vapnik, *Statistical learning theory*. Wiley New York, 1998, vol. 1.

[43] T. L. Griffiths, C. Kemp, and J. B. Tenenbaum, "Bayesian models of cognition," 2008.

[44] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95, Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1104–1111, ISBN: 1-55860-363-8.

[45] S. Ekvall and D. Kragic, "Robot learning from demonstration: A task-level planning approach," *International Journal of Advanced Robotic Systems*, vol. 5, no. 3, p. 33, 2008.

[46] J. Bentham, *An Introduction to the Principles of Morals and Legislation*. 1789.

[47] M. L. Puterman, "Markov decision process," *Journal of the Royal Statistical Society*, vol. 158, no. 3, pp. 1–16, 1994.

[48] Y. Zhu, C. Jiang, Y. Zhao, D. Terzopoulos, and S.-C. Zhu, "Inferring forces and learning human utilities from videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3823–3833.

[49] S. R. Andrew Y. Ng, "Algorithms for inverse reinforcement learning," in *Proceedings of International Conference on Machine Learning (ICML 2000)*, Stanford, USA, 2000.

[50] D. Hadfieldmenell, A. Dragan, P. Abbeel, and S. Russell, "Cooperative inverse reinforcement learning," 2016.

[51] A. Gabaldon, "Non-markovian control in the situation calculus," *Artificial Intelligence*, vol. 175, no. 1, pp. 25–48, 2011.

[52] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza, "Decision-theoretic planning with non-markovian rewards," *Journal of Artificial Intelligence Research*, vol. 25, pp. 17–74, 2006.

[53] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[54] P. Wei, Y. Zhao, N. Zheng, and S.-C. Zhu, "Modeling 4d human-object interactions for event and object recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3272–3279.

[55] E. T. Mueller, *Commonsense reasoning: an event calculus based approach*. Morgan Kaufmann, 2014.

[56] S. C. Zhu and D. Mumford, "Prior learning and gibbs reaction-diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 11, pp. 1236–1250, 1997.

[57] I. Caragiannis, S. Nath, A. D. Procaccia, and N. Shah, "Subset selection via implicit utilitarian voting," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 151–157.

[58] K Salkauskas, "C1 splines for interpolation of rapidly varying data," *Rocky Mountain Journal of Mathematics*, vol. 14, no. 1, 1974.

[59] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 1168–1175.

[60] D. Xie, S. Todorovic, and S.-C. Zhu, "Inferring dark matteri and dark energyi from videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2224–2231.

[61] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[62] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.

[63] A. Sen, "Social choice theory," *Handbook of mathematical economics*, vol. 3, pp. 1073–1181, 1986.

[64] X. Jiang, L.-H. Lim, Y. Yao, and Y. Ye, "Statistical ranking and combinatorial hodge theory," *Mathematical Programming*, vol. 127, no. 1, pp. 203–244, 2011.

[65] N. Shukla, *Machine Learning with TensorFlow*. Manning Publications, 2017, ISBN: 1617293873.

[66] D. P. Bertsekas, "Dynamic programming and stochastic control," *Mathematics in science and engineering*, vol. 125, pp. 222–293, 1976.

[67] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[68] J. McCarthy and P. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in *Readings in Artificial Intelligence*, B. L. Webber and N. J. Nilsson, Eds., Morgan Kaufmann, 1981, pp. 431 –450, ISBN: 978-0-934613-03-3.

[69] R. Reiter, *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press, 2001.

[70] C. H. Reinsch, "Smoothing by spline functions," *Numerische mathematik*, vol. 10, no. 3, pp. 177–183, 1967.

[71] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005, pp. 89–96.

[72] S. Park, B. Xiaohan Nie, and S.-C. Zhu, "Attribute And-Or Grammar for Joint Parsing of Human Attributes, Part and Pose," *ArXiv e-prints*, May 2016. arXiv: 1605.02112 [cs.CV].

[73] M. Cook, "Universality in elementary cellular automata," *Complex systems*, vol. 15, no. 1, pp. 1–40, 2004.

[74] N. Shukla, Y. He, F. Chen, and S. Zhu, "Learning human utility from video demonstrations for deductive planning in robotics," in *1st Annual Conference on Robot Learning (CoRL 2017)*, 2017.

[75] R. E. Bellman, *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[76] N. Shukla, N. Solano, and V. Zhang, "Dialogue manager programming language (dmpl)," W3C, Tech. Rep., Apr. 2019, https://www.w3.org/2019/04/dmpl/.

[77] Y. Cao, D. Barrett, A. Barbu, S. Narayanaswamy, H. Yu, A. Michaux, Y. Lin, S. Dickinson, J. Mark Siskind, and S. Wang, "Recognize human activities from partially observed videos," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[78] J. M. Siskind, "Visual event classification via force dynamics," 2000.

[79] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," 2008.

[80] K. Li and J. W. Burdick, "Inverse reinforcement learning in large state spaces via function approximation," *CoRR*, vol. abs/1707.09394, 2017.

[81] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, p. 1.

[82] F. Morbini, D. DeVault, K. Sagae, J. Gerten, A. Nazarian, and D. Traum, "Flores: A forward looking, reward seeking, dialogue manager," pp. 313–325, 2014.

[83] A. Gogol-Döring, D. Weese, T. Rausch, and K. Reinert, "Seqan an efficient, generic c++ library for sequence analysis," *BMC Bioinformatics*, vol. 9, pp. 11 –11, 2007.

[84] R. A. Brooks, "Symbolic error analysis and robot planning," *The International Journal of Robotics Research*, vol. 1, no. 4, pp. 29–78, 1982.

[85] M. D. Escobar and M. West, "Bayesian density estimation and inference using mixtures," *Journal of the american statistical association*, vol. 90, no. 430, pp. 577–588, 1995.

[86] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[87] A. Veit, B. Kovacs, S. Bell, J. McAuley, K. Bala, and S. Belongie, "Learning visual clothing style with heterogeneous dyadic co-occurrences," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4642–4650.

[88] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[89] H. L. Younes and M. L. Littman, "Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects," 2004.

[90] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of statistical software*, vol. 76, no. 1, 2017.

[91] A. Pfeffer, "Ibal: A probabilistic rational programming language," Citeseer.

[92] A. Nath and P. Domingos, "A language for relational decision theory."

[93] G. Van den Broeck, I. Thon, M. Van Otterlo, and L. De Raedt, "Dtproblog: A decision-theoretic probabilistic prolog," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[94] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebric decision diagrams and their applications," *Formal methods in system design*, vol. 10, no. 2-3, pp. 171–206, 1997.

[95] C. Boutilier, R. Reiter, M. Soutchanski, S. Thrun, *et al.*, "Decision-theoretic, high-level agent programming in the situation calculus," 2000.