Learning Parameterized Task Structure for Generalization to Unseen Entities

Anthony Liu*[1], Sungryull Sohn*[2], Mahdi Qazwini[1] , Honglak Lee[1,2]

[1] University of Michigan
[2] LG AI Research
anthliu@umich.edu, srsohn@lgresearch.ai, mqazwini@umich.edu, honglak@eecs.umich.edu

Abstract

Real world tasks are hierarchical and compositional. Tasks can be composed of multiple subtasks (or sub-goals) that are dependent on each other. These subtasks are defined in terms of entities (e.g., `apple, pear`) that can be recombined to form new subtasks (e.g., `pickup apple`, and `pickup pear`). To solve these tasks efficiently, an agent must infer subtask dependencies (e.g. an agent must execute `pickup apple` before `place apple in pot`), and generalize the inferred dependencies to new subtasks (e.g. `place apple in pot` is similar to `place apple in pan`). Moreover, an agent may also need to solve unseen tasks, which can involve unseen entities. To this end, we formulate parameterized subtask graph inference (PSGI), a method for modeling subtask dependencies using first-order logic with subtask entities. To facilitate this, we learn entity attributes in a zero-shot manner, which are used as quantifiers (e.g. `is_pickable(X)`) for the parameterized subtask graph. We show this approach accurately learns the latent structure on hierarchical and compositional tasks more efficiently than prior work, and show PSGI can generalize by modelling structure on subtasks unseen during adaptation.

1 Introduction

Real world tasks are *hierarchical*. Hierarchical tasks are composed of multiple sub-goals that must be completed in certain order. For example, the cooking task shown in Figure 1 requires an agent to boil some food object (e.g. `Cooked egg`). An agent must place the food object x in a cookware object y, place the cookware object on the stove, before boiling this food object x. Parts of this task can be decomposed into sub-goals, or *subtasks* (e.g. `Pickup egg`, `Put egg on pot`). Solving these tasks requires long horizon planning and reasoning ability (Erol 1996; Xu et al. 2018; Ghazanfari and Taylor 2017; Sohn, Oh, and Lee 2018). This problem is made more difficult of rewards are *sparse*, if only few of the subtasks in the environment provide reward to the agent.

Real world tasks are also *compositional* (Carvalho et al. 2020; Loula, Baroni, and Lake 2018; Andreas, Klein, and Levine 2017; Oh et al. 2017). Compositional tasks are often made of different "components" that can recombined to form new tasks. These components can be numerous, leading to a *combinatorial* number of subtasks. For example, the cooking task shown in Figure 1 contains subtasks that follow a verb-objects structure. The verb `Pickup` admits many subtasks, where any object x composes into a new subtask (e.g. `Pickup egg`, `Pickup pot`). Solving compositional tasks also requires reasoning (Andreas, Klein, and Levine 2017; Oh et al. 2017). Without reasoning on the relations between components between tasks, exploring the space of a combinatorial number of subtasks is extremely inefficient.

In this work, we propose to tackle the problem of hierarchical *and* compositional tasks. Prior work has tackled learning hierarchical task structures by modelling dependencies between subtasks in a *graph structure* (Sohn, Oh, and Lee 2018; Sohn et al. 2020; Xu et al. 2018; Huang et al. 2019). In these settings, during training, the agent tries to efficiently adapt to a task by inferring the latent graph structure, then uses the inferred graph to maximize reward during test. However, this approach does not scale for compositional tasks. Prior work tries to infer the structure of subtasks *individiually* – they do not consider the relations between compositional tasks.

We propose the *parameterized subtask graph inference* (PSGI) approach for tackling hierarchical and compositional tasks. We present an overview of our approach in Figure 1. This approach extends the problem introduced by (Sohn et al. 2020). Similar to (Sohn et al. 2020), we assume options (Sutton, Precup, and Singh 1999) (low level policies) for completing subtasks have been trained or are given as subroutines for the agent. These options are imperfect, and require certain conditions on the state to be meet before they can be successfully executed. We model the problem as a transfer RL problem. During training, an exploration policy gathers trajectories. These trajectories are then used to infer the latent *parameterized subtask graph*, $\widehat{\mathcal{G}}$. $\widehat{\mathcal{G}}$ models the hierarchies between compositional tasks and options in symbolic graph structure (shown in 1). In PSGI, we infer the *preconditions* of options, subtasks that must be completed before an option can be successfully executed, and the *effects* of options, subtasks that are completed after they are executed. The parameterized subtask graph is then used to maximize reward in the test environment by using GRProp, a method introduced by (Sohn, Oh, and Lee 2018) which propagates a gradient through $\widehat{\mathcal{G}}$ to learn the test policy.

In PSGI, we use *parameterized* options and subtasks. This

*These authors contributed equally.

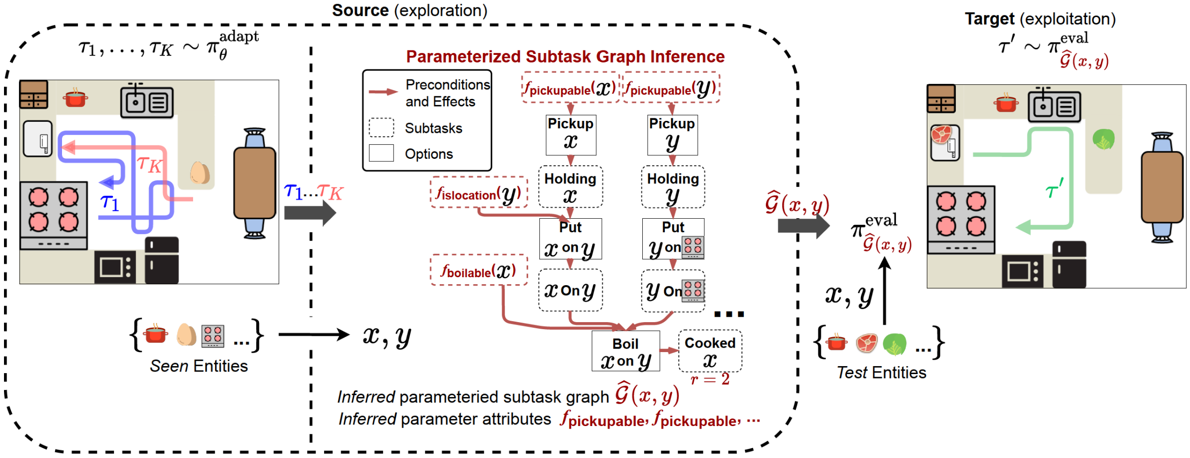

Figure 1: We present an overview of *parameterized subtask graph inference* (PSGI) in a toy cooking environment. In various tasks, the agent must cook various foods to receive reward. *Left:* The adaptation policy $\pi_\theta^{\text{adapt}}$ initially explores the cooking source task (training), generating a trajectories $\tau_1, \ldots, \tau_K$. *Middle:* Using $\tau_1, \ldots, \tau_K$, the agent infers a *parameterized subtask graph* $\widehat{\mathcal{G}}$ of the environment, which describes the preconditions and effects between *parameterized* options and subtasks using (x and y) over *entities* (objects in the environment). The agent learns a set of parameter attributes ($\widehat{A}_{\text{att}} = f_{\text{pickupable}} \ldots$) in a *zero-shot* manner and uses these attributes to construct $\widehat{\mathcal{G}}$. *Right:* The agent initializes a separate test policy $\pi^{\text{test}}_{\widehat{\mathcal{G}}}$ that maximizes reward by following the inferred parameterized subtask graph $\widehat{\mathcal{G}}$. In this target environment (test) there exist *unseen parameters* (cabbage and meat). Preconditions and effects for these parameters are accurately inferred by substituting for entities (x and y).

allows PSGI to infer the latent task structure in a *first-order logic* manner. For example, in the cooking task in Figure 1 we represent all `Pickup`-object options using a *paramaterized option*, `Pickup` x. Representing options and subtasks in parameterized form serves two roles: 1. The resulting graph is more **compact**. There is less redundancy when representing compositional tasks that share common structure. Hence a parameterized subtask graph requires less samples to infer (e.g. relations for `Pickup apple`, `Pickup pan`, etc. are inferred at once with `Pickup` x). 2. The resulting graph can **generalize** to *unseen* subtasks, where unseen subtasks may share similar structure but are not encountered during adaptation (e.g. `Pickup cabbage` in Figure 1).

To enable parameterized representation, we also learn the *attributes* of the components in compositional tasks. These attributes are used to differentiate structures of parameterized options and subtasks. For example, in the cooking task Figure 1, not every object can be picked up with `Pickup`, so the inferred attribute $\widehat{f}_{\text{pickupable}}(x)$ is a precondition to `Pickup`(x). Similarly, in a more complex cooking task, some object x may need to be sliced, before it can be boiled (e.g. cabbage), but some do not (e.g. egg). We model these structures using *parameter attributes*, $\widehat{A}_{\text{att}}$ (in the cooking task case objects are parameters). In this work we present a simple scheme to infer attributes in a *zero-shot* manner. These attributes are then used to generalize to other parameters (or entities), that may be unseen during adaptation.

We summarize our work as follows:

- We propose the approach of *parameterized subtask graph inference* (PSGI) to efficiently infer the subtask structure of hierarchical and compositional tasks in a **first order logic manner**.
- We propose a simple *zero-shot* learning scheme to infer *entity attributes*, which are used to relate the structures of compositional subtasks.
- We demonstrate PSGI on a symbolic cooking environment that has complex hierarchical and compositional task structure. We show PSGI can accurately infer this structure more *efficiently* than prior work and *generalize* this structure to unseen tasks.

2 Problem Definition

Background: Transfer Reinforcement Learning

A *task* is characterized by an MDP $\mathcal{M}_G = \langle \mathcal{A}, \mathcal{S}, \mathcal{T}_G, \mathcal{R}_G \rangle$, which is parameterized by a task-specific G, with an action space $\mathcal{A}$, state space $\mathcal{S}$, transition dynamics $\mathcal{T}_G$, and reward function $\mathcal{R}_G$. In the transfer RL formulation (Duan et al. 2016; Finn, Abbeel, and Levine 2017), an agent is given a fixed distribution of training tasks $\mathcal{M}^{\text{train}}$, and must learn to efficiently solve a distribution of unseen test tasks $\mathcal{M}^{\text{test}}$. Although these distributions are disjoint, we assume there is some similarity between tasks such that some learned behavior in training tasks may be useful for learning test tasks. In each task, the agent is given k timesteps to interact with the environment (the *adaptation* phase), in order to adapt to the given task. After, the agent is evaluated on its adaptation (the *test* phase). The agent's performance is measured in terms of the expected return:

$$\mathcal{R}_{\mathcal{M}_G} = \mathbb{E}_{\pi_k, \mathcal{M}_G}\left[\sum_{t=1}^{H} r_t\right] \tag{1}$$

where π_K is the policy after k timesteps of the adaptation phase, H is the horizon in the test phase, and r_t is the reward at time t of the test phase.

Background: The Subtask Graph Problem

The subtask graph inference problem is a transfer RL problem where tasks are parameterized by hierarchies of *subtasks* (Sohn et al. 2020), Φ. A task is composed of N subtasks, $\{\phi^1, \dots, \phi^N\} \subset \Phi$, where each subtask $\phi \in \Phi$ is parameterized by the tuple $\langle \mathcal{S}_{\text{comp}}, G_r \rangle$, a *completion set* $\mathcal{S}_{\text{comp}} \subset \mathcal{S}$, and a *subtask reward* $G_r : \mathcal{S} \to \mathbb{R}$. The completion set $\mathcal{S}_{\text{comp}}$ denotes whether the subtask ϕ is *complete*, and the subtask reward G_r is the reward given to the agent when it completes the subtask.

Following (Sohn et al. 2020), we assume the agent learns a set of options $\mathbb{O} = \{\mathcal{O}^1, \mathcal{O}^2, \dots\}$ that *completes* the corresponding subtasks (Sutton, Precup, and Singh 1999). These options can be learned by conditioning on subtask goal reaching reward: $r_t = \mathbb{I}(s_t \in \mathcal{S}^i_{\text{comp}})$. Each option $\mathcal{O} \in \mathbb{O}$ is parameterized by the tuple $\langle \pi, G_{\text{prec}}, G_{\text{effect}} \rangle$. There is a trained policy π corresponding to each $\mathcal{O}$. These options may be *eligible* at different precondition states $G_{\text{prec}} \subset \mathcal{S}$, where the agent must be in certain states when executing the option, or the policy π fails to execute (also the *initial set* of $\mathcal{O}$ following (Sutton, Precup, and Singh 1999)). However, unlike (Sohn et al. 2020), these options may complete an unknown number of subtasks (and even *remove* subtask completion). This is parameterized by $G_{\text{effect}} \subset \mathcal{S}$ (also the *termination set* of $\mathcal{O}$ following (Sutton, Precup, and Singh 1999)).

Environment: We assume that the subtask completion and option eligibility is known to the agent. (But the precondition, effect, and reward is hidden and must be inferred). In each timestep t the agent is the state $s_t = \{x_t, e_t, \text{step}_t, \text{step}_{\text{phase},t}, \text{obs}_t\}$.

- **Completion:** $x_t \in \{0,1\}^N$ denotes which subtasks are complete.
- **Eligibility:** $e_t \in \{0,1\}^M$ denotes which options are eligible.
- **Time Budget:** $\text{step}_t \in \mathbb{Z}_{>0}$ is the number steps remaining in the episode.
- **Adaptation Budget:** $\text{step}_{\text{phase},t} \in \mathbb{Z}_{>0}$ is the number steps remaining in the adaptation phase.
- **Observation:** $\text{obs}_t \in \mathbb{R}^d$ is a low level observation of the environment at time t.

The Parameterized Subtask Graph Problem

Subtasks and Option Entities In the real world, compositional subtasks can be described in terms of a set of *entities*, $\mathcal{E}$. (e.g. `pickup`, `apple`, `pear`, $\dots \in \mathcal{E}$) that can be recombined to form new subtasks (e.g. (`pickup`, `apple`), and (`pickup`, `pear`)). We assume that these entities are given to the agent. Similarly, the learned options that execute these subtasks can also be parameterized by the same entities (e.g. [`pickup`, `apple`], and [`pickup`, `pear`]).

In real world tasks, we expect options with entities that share "attributes" to have similar policy, precondition, and effect. For example, options [`cook`, `egg`, `pot`] and [`cook`, `cabbage`, `pot`] share similar preconditions (the target ingredient must be placed in the *pot*), but also different (*cabbage* must be sliced, but the egg does not). In this example, *egg* and *cabbage* are both *boilable*, but *egg* is not *sliceable*.

To model these similarities, we assume in each task, there exist boolean *latent attribute functions* which indicate shared attributes in entities. E.g. $f_{\text{pickupable}} : \mathcal{E} \to \{0,1\}$, where $f_{\text{pickupable}}(\texttt{apple}) = 1$. We will later try to infer the values of these latent entities, so we additionally assume there exist some weak supervision, where a low-level embedding of entities is provided to the agent, $f_{\text{entityembed}} : \mathcal{E} \to \mathbb{R}^D$.

The Parameterized Subtask Graph Our goal is to infer the underlying task structure between subtasks and options so that the agent may complete subtasks in an optimal order. As defined in the previous sections, this task structure can be completely determined by the option preconditions, option effects, and subtask rewards. We define the *parameterized subtask graph* to be the tuple of the *parameterized* preconditions, effects, and rewards for all subtasks and options:

$$\mathcal{G} = \langle \mathcal{G}_{\text{prec}}, \mathcal{G}_{\text{eff}}, \mathcal{G}_r \rangle \tag{2}$$

where $\mathcal{G}_{\text{prec}} : \mathcal{E}^N \times \mathcal{S} \to \{0,1\}, \mathcal{G}_{\text{eff}} : \mathcal{E}^N \times \mathcal{S} \to \mathcal{S}$, and $\mathcal{G}_r : \mathcal{E}^N \times \mathcal{S} \to \mathbb{R}$. The parameterized precondition, $\mathcal{G}_{\text{prec}}$, is a function from an option with N entities and a subtask completion set to $\{0,1\}$, which specifies whether the option is eligible under a completion set. E.g. If $\mathcal{G}_{\text{prec}}([X_1, X_2], s) = 1$, then the option $[X_1, X_2]$ is eligible if the agent is in state s. The parameterized effect, $\mathcal{G}_{\text{eff}}$, is a function from an option with N entities and subtask completion set to a different completion set. Finally, the parameterized reward, $\mathcal{G}_r$, is a function from a subtask with N entities to the reward given to the agent from executing that subtask.

Our previous assumption that options with similar entities and attributes share preconditions and effects manifests in $\mathcal{G}_{\text{prec}}$ and $\mathcal{G}_{\text{eff}}$ where these functions tend to be *smooth*. Similar inputs to the function (similar option entities) tend to yield similar output (similar eligibility and effect values). This smoothness gives two benefits. 1. We can share experience between similar options for inferring preconditions and effect. 2. This enables generalization to preconditions and effects of unseen entities. Note that this smoothness does *not* apply to the reward $\mathcal{G}_r$. We assume reward given for subtask completion is independent across tasks.

3 Method

We propose the *Parameterized Subtask Graph Inference* (PSGI) method to efficiently infer the latent parameterized subtask graph $\mathcal{G} = \langle \mathcal{G}_{\text{prec}}, \mathcal{G}_{\text{eff}}, \mathcal{G}_r \rangle$. Figure 2 gives an overview of our approach[1]. At a high level, we use the adaptation phase to gather *adaptation trajectories* from the environment using an adaptation policy $\pi^{\text{adapt}}_\theta$. Then, we use the adaptation trajectories to infer the latent subtask graph $\widehat{\mathcal{G}}$. In the test phase, a *test policy* $\pi^{\text{test}}_{\widehat{\mathcal{G}}}$ is conditioned on the

[1]An implementation of PSGI and experiments is available at https://github.com/anthliu/PSGI

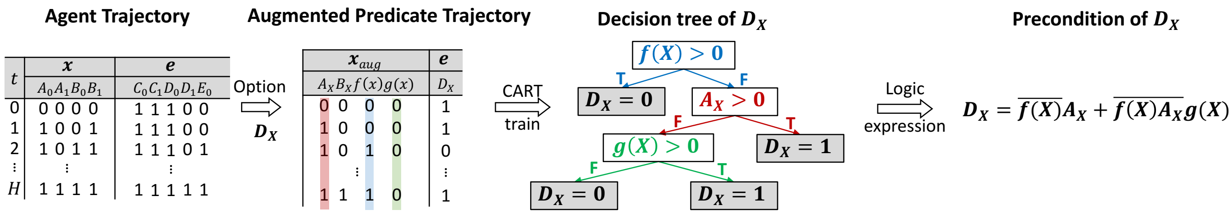

Figure 2: An overview of our approach for estimating the parameterized precondition of the subtask graph $\widehat{\mathcal{G}_{\text{prec}}}$ in a simple environment with subtasks A, B and options C, D, E. Each subtask and option has a parameter 0 or 1. Note by inferring the parameterized precondition and effects, we can infer the behavior unseen subtasks and options such as D_2. We run precondition inference for every option and show D_X as an example. 1. The first table is built from the agent's trajectory (x is the subtask completion, e the option eligibility). 2. We build the second table, the "augmented" trajectory by substituting X into all possible subtask completions, A_X, B_X, and inferred attributes f, g. 3. We train a decision tree over the table, to infer the relation $x_{\text{aug}} \rightarrow D_x$ (predicting when D_x is eligible given the completion x_{aug}). 4. We translate the decision tree into an equivalent predicate boolean expression, which is one part of the inferred parameratized subtask graph $\widehat{\mathcal{G}}$.

inferred subtask graph $\widehat{\mathcal{G}}$ and maximizes the reward. As the performance of the test policy is dependent on the inferred subtask graph $\widehat{\mathcal{G}}$, it is important to accurately infer this graph. Note that the test task may contain subtasks that are *unseen* in the training task. We learn a predicate subtask graph $\widehat{\mathcal{G}}$ that can *generalize* to these unseen subtasks and options.

Zero-shot Learning Entity Attributes

In the *Parameterized Subtask Graph Problem* definition, we assume there exist *latent attributes* that indicate shared structure between options and subtasks with the same attributes. E.g. One attribute may be $f_{\text{pickupable}} : \mathcal{E} \rightarrow \{0, 1\}$, where $f_{\text{pickupable}}(\texttt{apple}) = 1$, etc. Our goal is to infer a set of candidate attribute functions, $\widehat{A}_{\text{att}} = \{\widehat{f}_1, \widehat{f}_2, \ldots\}$, such that options with the same attributes indicates the same preconditions. As there is no supervision involved, we formulate this inference as a *zero shot learning problem* (Palatucci et al. 2009). Note the inferred attributes that are preconditions for options should also construct an accurate predicate subtask graph for options *unseen* in the adaptation phase.

During the adaptation phase, the agent will encounter a set of seen entities $E \subset \mathcal{E}$. We construct candidate attributes from E using our *smoothness* assumption, where similar entities result in similar preconditions. We generate candidate attributes based on similarity using the given entity embedding, $f_{\text{entityembed}} : \mathcal{E} \rightarrow \mathbb{R}^D$.

Let $C = \{C_1, C_2, \ldots\}$ be an exhaustive set of clusters generated from E using $f_{\text{entityembed}}$. Then, we define a candidate attribute function from each cluster: $\widehat{f}_i(X) := \mathbb{I}[X \in C_i]$ To infer the attribute of an unseen entity $X \notin E$, we use a 1-Nearest Neighbor classifier that uses the attributes of the nearest seen entity (Fix 1985). $\widehat{f}_i(X) = \mathbb{I}[X^* \in C_i]$ where $X^* = \text{argmin}_{X' \in E} \text{dist}(f_{\text{entityembed}}(X), f_{\text{entityembed}}(X'))$.

Parameterized Subtask Graph Inference

Let $\tau_H = \{s_1, o_1, r_1, d_1, \ldots, s_H\}$ be the adaptation trajectory of the adaptation policy $\pi_\theta^{\text{adapt}}$ after H time steps. Our goal is to infer the maximum likelihood parameterized subtask graph $\mathcal{G}$ given this trajectory τ_H.

$$\widehat{\mathcal{G}}^{\text{MLE}} = \underset{\mathcal{G}_{\text{prec}}, \mathcal{G}_{\text{eff}}, \mathcal{G}_r}{\text{argmax}} \; p(\tau_H | \mathcal{G}_{\text{prec}}, \mathcal{G}_{\text{eff}}, \mathcal{G}_r) \tag{3}$$

By expanding this likelihood term, we show that to maximize $\widehat{\mathcal{G}}$, it suffices to maximize $\widehat{\mathcal{G}}_{\text{prec}}$, $\widehat{\mathcal{G}}_{\text{eff}}$, and $\widehat{\mathcal{G}}_r$ individually.

$$\widehat{\mathcal{G}}^{\text{MLE}} = \left(\widehat{\mathcal{G}}^{\text{MLE}}_{\text{prec}}, \widehat{\mathcal{G}}^{\text{MLE}}_{\text{eff}}, \widehat{\mathcal{G}}^{\text{MLE}}_r\right) \tag{4}$$

$$= \Bigg(\underset{\mathcal{G}_{\text{prec}}}{\text{argmax}} \prod_{t=1}^{H} p(e_t | x_t, \mathcal{G}_{\text{prec}}), \tag{5}$$

$$\underset{\mathcal{G}_{\text{eff}}}{\text{argmax}} \prod_{t=1}^{H} p(x_{t+1} | x_t, o_t, \mathcal{G}_{\text{eff}}), \tag{6}$$

$$\underset{\mathcal{G}_r}{\text{argmax}} \prod_{t=1}^{H} p(r_t | o_t, o_{t+1}, \mathcal{G}_r) \Bigg) \tag{7}$$

We show details of this derivation in the appendix. Next, we explain how to compute $\widehat{\mathcal{G}}_{\text{prec}}$, $\widehat{\mathcal{G}}_{\text{eff}}$, and $\widehat{\mathcal{G}}_r$.

Parameterized Precondition Inference via Predicate Logic Induction We give an overview of how we infer the option preconditions $\widehat{\mathcal{G}}_{\text{prec}}$ in Figure 2. Note from the definition, we can view the precondition $\mathcal{G}_{\text{prec}}$ as a deterministic function, $f_{\mathcal{G}_{\text{prec}}} : (E, x) \mapsto \{0, 1\}$, where E is the option entities, and x is the completion set vector. Hence, the probability term in Eq.(6) can be written as $p(e_t | x_t, \mathcal{G}_{\text{prec}}) = \prod_{i=1}^{N} \mathbb{I}[e_t^{(i)} = f_{\mathcal{G}_{\text{prec}}}(E^{(i)}, x_t)]$ where $\mathbb{I}$ is the indicator function, and $E^{(i)}$ is the entity set of the ith option in the given task. Thus, we have

$$\widehat{\mathcal{G}}^{\text{MLE}}_{\text{prec}} = \underset{\mathcal{G}_{\text{prec}}}{\text{argmax}} \prod_{t=1}^{H} \prod_{i=1}^{N} \mathbb{I}[e_t^{(i)} = f_{\mathcal{G}_{\text{prec}}}(E^{(i)}, x_t)] \tag{8}$$

Following (Sohn et al. 2020), this can be maximized by finding a boolean function $\widehat{f}_{\mathcal{G}_{\text{prec}}}$ over *only* subtask completions x_t that satisfies all the indicator functions in Eq.(8). However this yields multiple possible solutions — particularly the preconditions of unseen option entities in the trajectory τ_H. If we infer a $\widehat{f}_{\mathcal{G}_{\text{prec}}}$ separately over all seen options (without considering the option parameters), this solution is identical to the solution proposed by (Sohn et al. 2020). We want to additionally generalize our solution over multiple unseen subtasks and options using the entities, E.

We leverage our smoothness assumption — that $\widehat{f}_{\mathcal{G}_{\text{prec}}}$ is *smooth* with respect to the input entities and attributes. E.g. If the inferred precondition for the option [pickup, X] is the candidate attribute $\widehat{f}(X)$, any entity X where $\widehat{f}(X) = 1$ has the same precondition. I.e. For some unseen entity set E^* we want the following property to hold:

$$\widehat{f}_i(E) = \widehat{f}_i(E^*) \text{ for some } i \;\Rightarrow \widehat{f_{\mathcal{G}_{\text{prec}}}}(E, x_t) = \widehat{f_{\mathcal{G}_{\text{prec}}}}(E^*, x_t) \tag{9}$$

To do this, we infer a boolean function $\widehat{f_{\mathcal{G}_{\text{prec}}}}$ over *both* subtask completions x_t *and* entity variables $X \in E$. We use (previously inferred) candidate attributes over entities, $\widehat{f}_i(X) \forall X \in E$ in the boolean function to serve as *quantifiers*. Inferring in this manner insures that the precondition function $\widehat{f}_{\mathcal{G}_{\text{prec}}}$ is *smooth* with respect to the input entities and attributes. Note that some but not all attributes may be shared in entities. E.g. [cook, cabbage] has similar but not the same preconditions as [cook, egg]. So, we cannot directly reuse the same preconditions for similar entities. We want to generalize between different *combinations* of attributes.

We translate this problem as an *inductive logic programming* (ILP) problem (Muggleton and De Raedt 1994). We infer the eligibility (boolean output) of some option $\mathcal{O}$ with some entities(s) $E = \{X_1, X_2, \dots\}$, from boolean input formed by all possible completion values $\{x_t^i\}_{t=1}^{H}$, and all attribute values $\{\widehat{f}_i(X)\}_{X \in E}^{i=1\dots}$. We use the *classification and regression tree* (CART) with Gini impurity to infer the the precondition functions $\widehat{f}_{\mathcal{G}_{\text{prec}}}$ for each parameter E (Breiman et al. 1984). Finally, the inferred decision tree is converted into an equivalent symbolic logic expression and used to build the parameterized subtask graph.

Parameterized Effect Inference We include an visualization of how we infer the option effects $\widehat{\mathcal{G}}_{\text{eff}}$ in the appendix in the interest of space. From the definitions of the parameterized subtask graph problem, we can write the predicate option effect $\mathcal{G}_{\text{eff}}$ as a deterministic function $f_{\mathcal{G}_{\text{eff}}} : (E, x_t) \mapsto x_{t+1}$, where if there is subtask completion x_t, executing option $\mathcal{O}$ (with entities E) successfully results in subtask completion x_{t+1}. Similar to precondition inference, we have

$$\widehat{\mathcal{G}}_{\text{eff}}^{\text{MLE}} = \operatorname*{argmax}_{\mathcal{G}_{\text{eff}}} \prod_{t=1}^{H} \prod_{i=1}^{N} \mathbb{I}[x_{t+1} = f_{\mathcal{G}_{\text{eff}}}(E^{(i)}, x_t)] \tag{10}$$

As this is deterministic, we can calculate the element-wise difference between x_t (before option) and x_{t+1} (after option) to infer $f_{\mathcal{G}_{\text{eff}}}$.

$$\widehat{f_{\mathcal{G}_{\text{eff}}}}(E^{(i)}, x) = x + \mathbb{E}_{t=1\dots H}[x_{t+1} - x_t | o_t = \mathcal{O}^{(i)}] \tag{11}$$

Similar to precondition inference, we also want to infer the effect of options with unseen parameters. We leverage the same smoothness assumption:

$$\widehat{f}_i(E) = \widehat{f}_i(E^*) \text{ for some } i \;\Rightarrow \widehat{f_{\mathcal{G}_{\text{eff}}}}(E, x_t) = \widehat{f_{\mathcal{G}_{\text{eff}}}}(E^*, x_t) \tag{12}$$

Unlike preconditions, we expect the effect function to be relatively constant across attributes, i.e., the effect of executing option [cook, X] is always completing the subtask (cooked, X), no matter the attributes of X. So we directly set the effect of unseen entities, $\widehat{f_{\mathcal{G}_{\text{eff}}}}(E^*, x_t)$, by similarity according to Equation 12.

Reward Inference We model the subtask reward as a Gaussian distribution $\mathcal{G}_r(E) \sim \mathcal{N}(\widehat{\mu}_E, \widehat{\sigma}_E)$. The MLE estimate of the subtask reward becomes the empirical mean of the rewards received during the adaptation phase when subtask with parameter $\mathcal{T}$ becomes complete. For the ith subtask in the task with entities E^i,

$$\widehat{\mathcal{G}}_r(E^i) = \widehat{\mu}_{E^i} = \mathbb{E}_{t=1\dots N}[r_t | x_{t+1}^i - x_t^i = 1] \tag{13}$$

Note we do not use the smoothness assumption for $\widehat{\mathcal{G}}_r(E)$, as we assume reward is independently distributed across tasks. We initialize $\widehat{\mathcal{G}}_r(E^*) = 0$ for unseen subtasks with entities E^* and update these estimates with further observation.

Task Transfer and Adaptation

In the test phase, we instantiate a *test policy* $\pi_{\widehat{\mathcal{G}}_{\text{prior}}}^{\text{test}}$ using the parameterized subtask graph $\widehat{\mathcal{G}}_{\text{prior}}$, inferred from the training task samples. The goal of the test policy is to maximize reward in the test environment using $\widehat{\mathcal{G}}_{\text{prior}}$. As we assume the reward is independent across tasks, we re-estimate the reward of the test task according to Equation 13, without task transfer. With the reward inferred, this yields the same problem setting given in (Sohn, Oh, and Lee 2018). (Sohn, Oh, and Lee 2018) tackle this problem using GRProp, which models the subtask graph as differentiable function over reward, so that the test policy has a dense signal on which options to execute are likely to maximally increase the reward.

However, the inferred parameterized subtask graph may be imperfect, the inferred precondition and effects may not transfer to the test task. To adapt to possibly new preconditions and effects, we use samples gathered in the adaptation phase of the test task to infer a new parameterized subtask graph $\widehat{\mathcal{G}}_{\text{test}}$, which we use to similarly instantiate another test policy $\pi_{\widehat{\mathcal{G}}_{\text{test}}}^{\text{test}}$ using GRProp. We expect $\widehat{\mathcal{G}}_{\text{test}}$ to eventually be more accurate than $\widehat{\mathcal{G}}_{\text{prior}}$ as more timesteps are gathered in the test environment. To maximize performance on test, we thus choose to instantiate a posterior test policy $\pi_{\text{posterior}}^{\text{test}}$, which is an *ensemble* policy over $\pi_{\widehat{\mathcal{G}}_{\text{prior}}}^{\text{test}}$ and $\pi_{\widehat{\mathcal{G}}_{\text{test}}}^{\text{test}}$. We heuristically set the weights of $\pi_{\text{posterior}}^{\text{test}}$ to favor $\pi_{\widehat{\mathcal{G}}_{\text{prior}}}^{\text{test}}$ early in the test phase, and $\pi_{\widehat{\mathcal{G}}_{\text{test}}}^{\text{test}}$ later in the test phase.

4 Related Work

Subtask Graph Inference. The subtask graph inference (SGI) framework (Sohn, Oh, and Lee 2018; Sohn et al. 2020) assumes that a task can be solved by completing a set of subtasks in the right order. SGI can efficiently solve these complex tasks by explicitly inferring the precondition relationship between subtasks in the form of a graph using an inductive logic programming (ILP) method. An execution policy (GrProp) uses the inferred graph to predict the optimal sequence of subtasks to be completed to solve the given task.

However, the proposed SGI framework is limited to a single task; the knowledge learned in one task cannot be transferred to another. This limits the SGI framework such that does not scale well to compositional tasks, and cannot generalize to unseen tasks. We extend the SGI framework by modeling *parameterized* subtasks and options, which encode relations between tasks to allow efficient and general learning. In addition, we generalize the SGI framework by learning an effect model – In the SGI framework it was assumed that for each subtask there is a corresponding option, that completes that subtask (and does not effect any other subtask).

Compositional Task Generalization. Prior work has also tackled compositional generalization in a symbolic manner (Loula, Baroni, and Lake 2018; Andreas, Klein, and Levine 2017; Oh et al. 2017). Loula, Baroni, and Lake (2018) test compositional generalization of natural language sentences in recurrent neural networks. Andreas, Klein, and Levine (2017); Oh et al. (2017) tackle compositional task generalization in an *instruction following* context, where an agent is given a natural language instruction describing the task the agent must complete (e.g. "pickup apple"). These works use *analogy making* to learn policies that can execute instructions by analogy (e.g. "pickup X"). However, these works construct policies on the *option level* – they construct policies that can execute "pickup X" on different X values. They also do not consider hierarchical structure for the order which options should be executed (as the option order is given in instruction). Our work aims to learn these analogy-like relations at a between-options level, where certain subtasks must be completed before another option can be executed.

Classical Planning. At a high level, a parameterized subtask graph $\mathcal{G}$ is similar to a STRIPS planning domain (Fikes and Nilsson 1971). Prior work in classical planning has proposed to learn STRIPS domain specifications through given trajectories (action traces) (Suárez-Hernández et al. 2020; Mehta, Tadepalli, and Fern 2011; Walsh and Littman 2008; Zhuo et al. 2010). Our work differs from these in 3 major ways: 1. PSGI learns an *attribute* model, crucial to generalizing compositional tasks. 2. We evaluate PSGI on more hierarchical domains, where prior work has evaluated on classical planning problems, which admit flat structure. 3. We evaluate PSGI on generalization, where there may exist subtasks and options that are not seen during adaptation.

5 Experiments

We aim to answer the following questions:

1. Can PSGI *generalize* to unseen evaluation tasks in zero-shot manner by transferring the inferred task structure?
2. Does PSGI *efficiently* infers the latent task structure compared to prior work (MSGI (Sohn et al. 2020))?

Environments

We evaluate PSGI in novel symbolic environments, **AI2Thor**, **Cooking**, and **Mining**. **AI2Thor** is a symbolic environment based on (Kolve et al. 2017), a simulated realistic indoor environment. In our **AI2Thor** environment, the agent is given a set of pre-trained options and must cook various food objects in different kitchen layouts, each containing possibly unseen objects. **Cooking** is a simplified cooking environment with similar but simpler dynamics to **AI2Thor**.The **Mining** domain is modelled after the open world video game Minecraft and the domain introduced by Sohn, Oh, and Lee (2018).

Tasks. In **AI2Thor**, there are 30 different tasks based on the 30 kitchen floorplans in (Kolve et al. 2017). In each task, 14 entities from the floorplan are sampled at random. Then, the subtasks and options are populated by replacing the parameters in parameterized subtasks and options by the sampled entities; ***e.g.***, we replace `X` and `Y` in the parameterized subtask `(pickup, X, Y)` by $\{$`apple, cabbage, table`$\}$ to populate nine subtasks. This results in 1764 options and 526 subtasks. The ground-truth attributes are taken from (Kolve et al. 2017) but are not available to the agent. **Cooking** is defined similarly and has a pool of 22 entities and 10 entities are chosen at random for each task. This results in 324 options and 108 subtasks. Similarly for **Mining**, we randomly sample 12 entities from a pool of 18 entities and populate 180 subtasks and 180 options for each task. In each environment, the reward is assigned at random to one of the subtasks that have the largest critical path length, where the critical path length is the minimum number of options to be executed to complete each subtask. See the appendix for more details on the tasks.

Observations. At each time step, the agent observes the completion and eligibility vectors (see section 2 for definitions) and the corresponding embeddings. The subtask and option embeddings are the concatenated vector of the embeddings of its entities; ***e.g.***, for `pickup, apple` the embedding is $[f(\texttt{pickup}), f(\texttt{apple})]$ where $f(\cdot)$ can be an image or language embeddings. In our experiments, we used 50 dimensional GloVE word embeddings (Pennington, Socher, and Manning 2014) as the embedding function $f(\cdot)$.

Baselines

- **MSGI$^+$** is the **MSGI** (Sohn et al. 2020) agent modified to be capable of solving our **Cooking** and **Mining** tasks. We augmented **MSGI** with an effect model, separate subtasks and options in the ILP algorithm, and replaced the GRProp with cyclic GRProp, a modified version of GRProp that can run with cycles in the subtask graph.
- **HRL** (Andreas, Klein, and Levine 2017) is the option-based hierarchical reinforcement learning agent. It is an actor-critic model over the pre-learned options.
- **Random** agent randomly executes any eligible option.

We meta-train **PSGI** on training tasks and meta-eval on evaluation tasks to test its adaptation efficiency and generalization ability. We train **HRL** on evaluation tasks to test its adaptation (*i.e.*, learning) efficiency. We evaluate **Random** baseline on evaluation tasks to get a reference performance. We use the same recurrent neural network with self-attention-mechanism so that the agent can handle varying number of (unseen) parameterized subtasks and options depending on the tasks. See the appendix for more details on the baselines.

Zero-shot Transfer Learning Performance

Figure 4 compares the zero-shot and few-shot transfer learning performance on **Cooking**, **AI2Thor**, and **Mining** do-

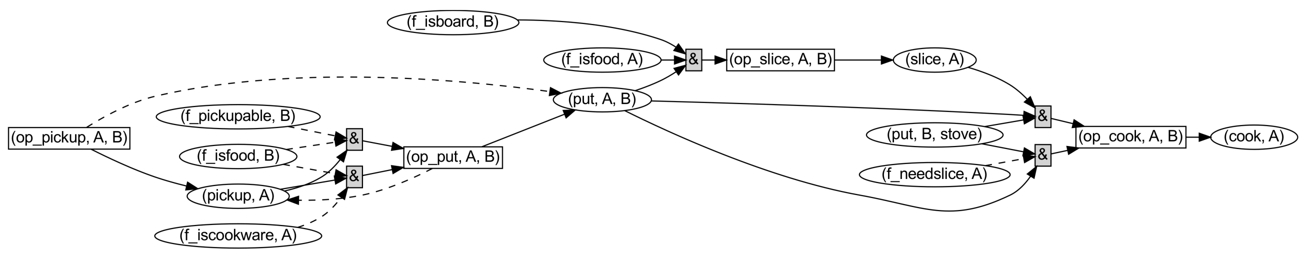

Figure 3: The inferred graph by PSGI after 2000 timesteps in the Cooking domain. Options are represented in rectangular nodes. Subtask completions and attributes are in oval nodes. A solid line represents a positive precondition / effect, dashed for negative.

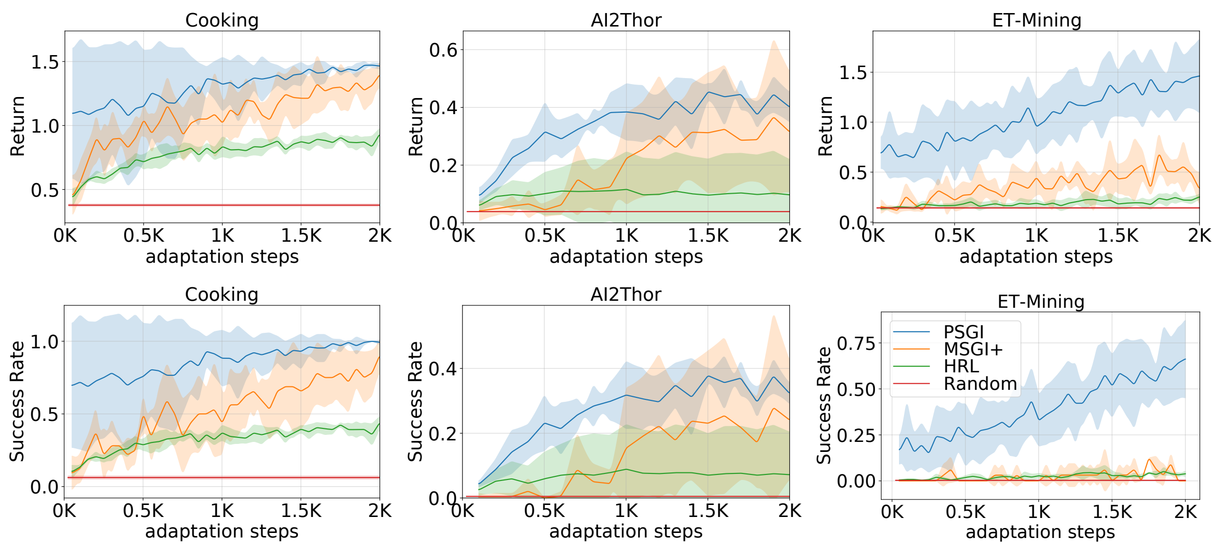

Figure 4: The adaptation curves in the Cooking, AI2Thor, and Mining domains.

mains. **PSGI** achieves over 50 and 20% success rate on **Cooking** and **Mining** domain without observing any samples in unseen evaluation tasks, which indicates **PSGI** generalizes well to unseen entities in zero-shot manner. Note that **MSGI**$^+$, **HRL**, and **Random** baselines have no ability to transfer its policy from training tasks to unseen evaluation tasks. **PSGI** achieves over 5% zero shot success rate on **AI2Thor**, still performing better than baselines, but relatively low compared to **PSGI** on **Cooking** and **Mining**, which indicates the high difficulty of transfer in **AI2Thor**.

Few-shot Transfer Learning Performance

In Figure 4, **PSGI** achieves high success rate after only 1000 steps of adaptation, while baselines do not learn any meaningful policy except **MSGI**$^+$ in **Cooking** and **AI2Thor**. This demonstrates the transfer learning abilities of **PSGI— PSGI** can share the experience of similar subtasks and options (*e.g.*, $\texttt{pickup}\ X\ \texttt{on}\ Y$ for all possible pairs of X and Y).

Comparison on Task Structure Inference

We ran **PSGI** and **MSGI**$^+$ in **Cooking**, **AI2Thor**, and **Mining**, inferring the latent subtask graphs for 2000 timesteps. The visualized inferred graphs at 2000 timesteps are shown in Figure 3. In the interest of space, we have shown the graph by **MSGI**$^+$ in the appendix. **PSGI** infers the parameterized graph using first-order logic, and thus it is more compact. However, **MSGI**$^+$ infers the subtask graph without parameterizing out the shared structure, resulting in a non-compact graph with hundreds of subtasks and options. Moreover, graph inferred by **PSGI** has 0% error in precondition and effect model inference. The graph inferred by **MSGI**$^+$ has 38% error in the preconditions (the six options that **MSGI**$^+$ completely failed to infer any precondition are not shown in the figure for readability).

6 Conclusion

In this work we presented *parameterized subtask graph inference* (PSGI), a method for efficiently inferring the latent structure of hierarchical and compositional tasks. PSGI also facilitates inference of *unseen* subtasks during adaptation, by inferring relations using predicates. PSGI additionally learns *parameter attributes* in a zero-shot manner, which differentiate the structures of different predicate subtasks. Our experimental results showed that PSGI is more efficient and more general than prior work. In future work, we aim to to tackle noisy settings, where options and subtasks exhibit possible failures, and settings where the option policies must also be learned.

Acknowledgements

The authors would like to thank Yiwei Yang and Wilka Carvalho for their valuable discussions. This work was supported in part by funding from LG AI Research.

References

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multi-task reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 166–175. PMLR.

Breiman, L.; Friedman, J.; Stone, C. J.; and Olshen, R. A. 1984. *Classification and regression trees*. CRC press.

Carvalho, W.; Liang, A.; Lee, K.; Sohn, S.; Lee, H.; Lewis, R. L.; and Singh, S. 2020. Reinforcement Learning for Sparse-Reward Object-Interaction Tasks in First-person Simulated 3D Environments. *arXiv preprint arXiv:2010.15195*.

Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P. L.; Sutskever, I.; and Abbeel, P. 2016. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Erol, K. 1996. *Hierarchical task network planning: formalization, analysis, and implementation*. Ph.D. thesis.

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

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 1126–1135. PMLR.

Fix, E. 1985. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine.

Ghazanfari, B.; and Taylor, M. E. 2017. Autonomous extracting a hierarchical structure of tasks in reinforcement learning and multi-task reinforcement learning. *arXiv preprint arXiv:1709.04579*.

Huang, D.-A.; Nair, S.; Xu, D.; Zhu, Y.; Garg, A.; Fei-Fei, L.; Savarese, S.; and Niebles, J. C. 2019. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8565–8574.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*.

Loula, J.; Baroni, M.; and Lake, B. M. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*.

Mehta, N.; Tadepalli, P.; and Fern, A. 2011. Autonomous learning of action models for planning. *Advances in Neural Information Processing Systems*, 24: 2465–2473.

Muggleton, S.; and De Raedt, L. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19: 629–679.

Oh, J.; Singh, S.; Lee, H.; and Kohli, P. 2017. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, 2661–2670. PMLR.

Palatucci, M.; Pomerleau, D.; Hinton, G. E.; and Mitchell, T. M. 2009. Zero-shot Learning with Semantic Output Codes. In *NIPS*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Sohn, S.; Oh, J.; and Lee, H. 2018. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *arXiv preprint arXiv:1807.07665*.

Sohn, S.; Woo, H.; Choi, J.; and Lee, H. 2020. Meta reinforcement learning with autonomous inference of subtask dependencies. *arXiv preprint arXiv:2001.00248*.

Suárez-Hernández, A.; Segovia-Aguas, J.; Torras, C.; and Alenyà, G. 2020. Strips action discovery. *arXiv preprint arXiv:2001.11457*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Walsh, T. J.; and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *AAAI*, volume 8, 714–719.

Xu, D.; Nair, S.; Zhu, Y.; Gao, J.; Garg, A.; Fei-Fei, L.; and Savarese, S. 2018. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 3795–3802. IEEE.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.