arXiv:2105.14074v3 [cs.AI] 30 Jun 2022

Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning

Rohan Chitnis*, **Tom Silver***, **Joshua B. Tenenbaum, Tomás Lozano-Pérez, Leslie Pack Kaelbling**
MIT Computer Science and Artificial Intelligence Laboratory
{ronuchit, tslvr, jbt, tlp, lpk}@mit.edu

***Abstract*— In robotic domains, learning and planning are complicated by continuous state spaces, continuous action spaces, and long task horizons. In this work, we address these challenges with Neuro-Symbolic Relational Transition Models (NSRTs), a novel class of models that are data-efficient to learn, compatible with powerful robotic planning methods, and generalizable over objects. NSRTs have both symbolic and neural components, enabling a bilevel planning scheme where symbolic AI planning in an outer loop guides continuous planning with neural models in an inner loop. Experiments in four robotic planning domains show that NSRTs can be learned very data-efficiently, and then used for fast planning in new tasks that require up to 60 actions and involve many more objects than were seen during training.**

I. Introduction

For robots to plan effectively, they will need to contend with continuous state spaces, continuous action spaces, and long task horizons (Figure 1, bottom row). Symbolic AI planning techniques are able to solve tasks with very long horizons, but typically assume discrete, factored spaces [3]. Neural network-based approaches have shown promise in continuous spaces, but scaling to long horizons remains challenging [4]. How can we combine symbolic and neural planning methods to overcome the limitations of each?

In this paper, we propose a new model-based approach for learning and planning in deterministic, goal-based, multi-task settings with continuous state and action spaces. Following previous work, we assume that a small number of discrete *predicates* (named relations over objects) are given, having been implemented by a human engineer [5], [6], or learned from previous experience in similar domains. These predicates induce discrete *state abstractions* of the continuous environment state. For example, HOLDING(`block1`) abstracts away the continuous pose with which `block1` is held. Even when given predicates, the question of *how to make use of them* to learn effective models for planning in continuous state and action spaces is a hard problem we seek to address.

From the predicates, and from training data of transitions in an environment, we aim to learn: (1) *abstract actions*, which define transitions between abstract states; (2) an *abstract transition model*, with symbolic preconditions and effects akin to AI planning operators; (3) a *neural transition model* over the low-level, continuous state and action spaces; and (4) a set of *neural action samplers*, which define how abstract actions can be refined into continuous actions.

*First two authors contributed equally.

We unify all of these with a new class of models that we term the **N**euro-**S**ymbolic **R**elational **T**ransition Model (NSRT) (pronounced "insert"). NSRTs have both symbolic and neural components; all components are relational, permitting generalization to tasks with any number of objects and allowing sample-efficient learning.

To plan with NSRTs, we borrow techniques from search-then-sample task and motion planning (TAMP) [7], with symbolic AI planning in an outer loop serving as guidance for continuous planning with neural models in an inner loop. This bilevel strategy allows for fast planning in continuous state and action spaces, while avoiding the *downward refinability assumption*, which would assume planning can be decomposed into separate symbolic and continuous planning steps [8]. When modeling robotic domains symbolically, the predicates are often *lossy*, meaning that downward refinability cannot be assumed (Figure 1, top and middle).

This paper focuses on *how to learn NSRTs* and *how to use NSRTs for planning* in continuous-space, long-horizon tasks. We show in four robotic planning domains, across both the PyBullet [2] and AI2-THOR [1] simulators, that NSRTs are extremely data-efficient: they can be learned from a few thousand transitions. We also show that learned NSRTs allow for fast planning on new tasks, with many more objects than during training and long horizons of up to 60 actions. Baseline and ablation comparisons confirm that integrated neuro-symbolic reasoning is key to these successes.

II. Related Work

Model-Based Reinforcement Learning (MBRL). Our work is related to MBRL in that we use data of taking actions in an environment to learn and plan with transition models. Many recent approaches to deep MBRL learn transition models that are relatively unstructured, and therefore must resort to undirected planning strategies such as CEM [4]. Relational MBRL is a subfield of MBRL that uses relational learning [9] to learn object-centric factored transition models [10] or to discover STRIPS operator models [11], [12] when given a set of predicates. Our work also learns relational transition models, but with a bilevel structure that allows planning without assuming downward refinability.

Symbolic AI Planning for RL. Our work continues a recent line of investigation that seeks to leverage symbolic AI planners for continuous states and actions. For example, previous work learns propositional [13], [14] or lifted [15],

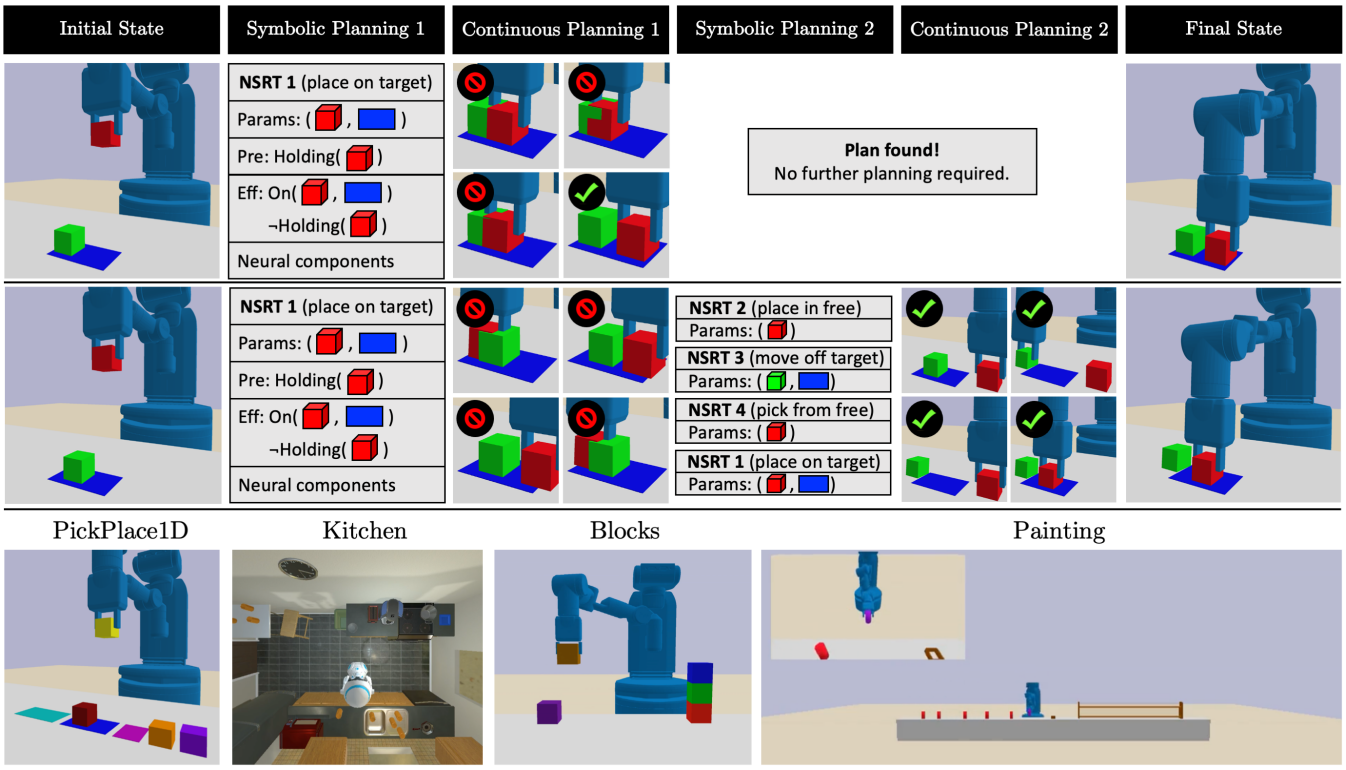

Fig. 1: We propose Neuro-Symbolic Relational Transition Models (NSRTs). (Top row) Given the goal of placing the red block completely into the blue target region, we first perform AI planning with the symbolic NSRT components to find a one-step symbolic plan. The Continuous Planning 1 column shows various ways in which the agent attempts to *refine* this one-step symbolic plan into a ground action, using the neural components of (ground) NSRT 1; it finds a collision-free refinement, shown in the Final State column. (Middle row) Here, the green block is initially in a slightly different position, so the red block has no room to be placed into the blue target region. The initial symbolic plan is the same. However, this symbolic plan is not *downward refinable*, so Continuous Planning 1 fails. The agent then continues on to consider a four-step symbolic plan that first moves the green object away (Symbolic Planning 2 column), which is successfully refined in the Continuous Planning 2 column. This example illustrates that in the presence of complex geometric constraints which make symbolic abstractions lossy, integrated symbolic and continuous reasoning is necessary. (Bottom row) Screenshots of our four robotic planning environments. Kitchen uses the AI2-THOR simulator [1]; the others use PyBullet [2].

[16], [17] symbolic transition models, and uses them with AI planners [18], [3]. Other related work has used symbolic planners as managers in hierarchical RL, where low-level option policies are learned [5], [19], [20]. This interface between symbolic planner and low-level policies assumes downward refinability, a critical assumption we do *not* make.

Learning for Hierarchical Planning. Reasoning at multiple levels of abstraction is a key theme in hierarchical planning [21]. Task and motion planners (TAMP) [7] can plan effectively at long horizons, but they typically require hand-specified operators, action samplers, and low-level transition models. Our work continues recent research into learning these components instead [6], [22], [23], [24]. Comparatively, ours is the first to learn operators, samplers, and a low-level transition model in one unified system.

III. Problem Setting

We study a deterministic, goal-based, multi-task setting with continuous object-oriented states, continuous actions, and a fixed, given set of predicates. Formally, we consider an *environment* $\langle \mathcal{T}, d, \mathcal{A}, f, \mathcal{P} \rangle$ and a collection of *tasks*, each of which is a tuple $\langle s_0, g, H \rangle$.

Environments. $\mathcal{T}$ is a set of object types, and $d : \mathcal{T} \to \mathbb{N}$ defines the dimensionality of the real-valued attribute (feature) vector of each object type. For example, an object of type `box` might have an attribute vector describing its current pose, side length, and color. An environment state s is a mapping from a set of typed objects o to attribute vectors of dimension $d(o)$, where $d(o)$ is shorthand for the dimension of the attribute vector of the type of object o. We use $\mathcal{S}$ to denote this object-oriented state space. The $\mathcal{A} \subseteq \mathbb{R}^m$ is the environment action space. The $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \cup \mathcal{S}_{\text{fail}}$ is a deterministic transition function mapping a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ to either a next state in $\mathcal{S}$ or a failure state in $\mathcal{S}_{\text{fail}}$. A failure state ends a task attempt, and is characterized by the objects responsible for the failure, e.g., objects that are unexpectedly in collision. *Throughout this paper, the transition function f is unknown to the agent.*

$\mathcal{P}$ is a set of *predicates* given to the agent. A predicate is a named, binary-valued relation among some number of objects. A *ground atom* applies a predicate to specific objects, such as $\textsc{Above}(o_1, o_2)$, where the predicate is $\textsc{Above}$. A *lifted atom* applies a predicate to typed placeholder variables: $\textsc{Above}(?a, ?b)$. Taken together, the set of

ground atoms that hold in a continuous state define a *discrete state abstraction*; let $\textsc{Abstract}(s)$ denote the abstract state for state $s \in \mathcal{S}$, and let $\mathcal{S}^\uparrow$ denote the abstract state space. For instance, a state s where objects o_1, o_2, and o_3 are stacked may be represented by the abstract state $\textsc{Abstract}(s) = \{\textsc{On}(o_1, o_2), \textsc{On}(o_2, o_3)\}$. Note that this abstract state loses details about the geometry of the scene.

Tasks. A task $\langle s_0, g, H \rangle$ is an initial state $s_0 \in \mathcal{S}$, a goal g, and a maximum horizon H. We will generally denote the set of objects in s_0 as $\mathcal{O}$. *This object set $\mathcal{O}$ is fixed within a task, but changes between tasks.* Goals g are sets of ground atoms over the object set $\mathcal{O}$, such as $\{\textsc{On}(o_3, o_2), \textsc{On}(o_2, o_1)\}$. A solution to a task is a *plan*, a sequence of at most H actions $a \in \mathcal{A}$ such that successive application of the unknown transition model f, starting from s_0, results in a final state s where $g \subseteq \textsc{Abstract}(s)$ (i.e., the goal holds).

Data Collection and Evaluation. We focus on the problems of *learning* and *planning*. To isolate these problems, we assume that a *training dataset* of tuples $\mathcal{D} = \{(s, a, s')\}$ is provided, with $s \in \mathcal{S}$, $a \in \mathcal{A}$, $s' \in \mathcal{S} \cup \mathcal{S}_{\text{fail}}$, and $f(s, a) = s'$. The agent's objective is to maximize the number of tasks solved over a set of *test tasks* that are selected to have long-horizon goals and more objects than were seen in $\mathcal{D}$.

IV. NSRT Representation

The next three sections introduce Neuro-Symbolic Relational Transition Models (NSRTs). In this section, we describe the NSRT representation; in Section V, we address planning with NSRTs; and in Section VI, we discuss learning NSRTs. Figure 2 illustrates the full pipeline.

We want models that are *learnable*, *plannable*, and *generalizable*. To that end, we propose the following definition:

Definition 1: A *Neuro-Symbolic Relational Transition Model (NSRT)* is a tuple $\langle O, P, E, h, \pi \rangle$, where:

- $O = (o_1, \ldots, o_k)$ is an ordered list of *parameters*; each o_i is a variable of some type from type set $\mathcal{T}$.
- P is a set of *symbolic preconditions*; each precondition is a lifted atom over parameters O.
- $E = (E^+, E^-)$ is a tuple of *symbolic effects*. E^+ are *add effects*, and E^- are *delete effects*; both are sets of lifted atoms over parameters O.
- $h : \mathbb{R}^{d(o_1)+\cdots+d(o_k)} \times \mathcal{A} \to \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$ is a *low-level transition model*, a neural network that predicts next attribute values given current ones and an action.
- $\pi(a \mid v)$ is an *action sampler*, a neural network defining a conditional distribution over actions $a \in \mathcal{A}$, where $v \in \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$ is a vector of attribute values.

In this paper, we will learn and plan with a *collection* of NSRTs. Together with the object set $\mathcal{O}$ of a task, a collection of NSRTs jointly defines four things: an *abstract action space* for efficient planning; a (partial) *abstract transition model* over the abstract state space $\mathcal{S}^\uparrow$ and the abstract action space; a (partial) *low-level transition model* over environment states and actions; and *action samplers* to refine abstract actions into environment actions. The rest of this section describes how NSRTs define these four components.

First, we define the notion of grounding an NSRT:

Definition 2: Given an object set $\mathcal{O}$, a *ground NSRT* is an NSRT whose parameters $o_i \in O$ are replaced by objects from $\mathcal{O}$, following an injective *substitution* σ mapping each o_i to an object. The ground preconditions and effects under σ are denoted P_σ and E_σ respectively.

Given a set of NSRTs and a task with object set $\mathcal{O}$, the resulting set of *ground* NSRTs defines an *abstract* action space for that task, which we denote as $\mathcal{A}^\uparrow$. Therefore, the phrases *abstract action* and *ground NSRT* are interchangeable. For instance, say we wrote an NSRT called Stack with two parameters $?x$ and $?y$; let $\sigma = \{?x \mapsto o_3, ?y \mapsto o_6\}$. Then $\textsc{Stack}(o_3, o_6)$ is an abstract action with substitution σ.

Working toward a definition of the abstract transition model, we next define ground NSRT *applicability*:

Definition 3: A ground NSRT with preconditions P_σ is *applicable* in state $s \in \mathcal{S}$ if $P_\sigma \subseteq \textsc{Abstract}(s)$. It is also *applicable* in abstract state $s^\uparrow \in \mathcal{S}^\uparrow$ if $P_\sigma \subseteq s^\uparrow$.

In words, applicability simply checks that the ground NSRT's precondition atoms are a subset of the abstract state atoms. A set of ground NSRTs defines a (partial) *abstract transition model* $f^\uparrow : \mathcal{S}^\uparrow \times \mathcal{A}^\uparrow \to \mathcal{S}^\uparrow$, which maps an abstract state and abstract action (ground NSRT) to a next abstract state. The $f^\uparrow(s^\uparrow, a^\uparrow)$ is partial since it is only defined when $a^\uparrow$ is applicable in $s^\uparrow$; when it *is* applicable, we have:

$$f^\uparrow(s^\uparrow, a^\uparrow) = (s^\uparrow \setminus E_\sigma^-) \cup E_\sigma^+, \qquad \text{(Equation 1)}$$

where $E_\sigma = (E_\sigma^+, E_\sigma^-)$ are the effects for $a^\uparrow$. In words, this abstract transition model removes delete effects and includes add effects, as long as the preconditions of the ground NSRT are satisfied. This symbolic representation is akin to operators in classical AI planning [25]; we use this connection to our advantage in Section V.

What is the connection between the symbolic components of an NSRT (P and E) and the environment transitions? To answer this question, we use the following definition:

Definition 4: A ground NSRT $a^\uparrow$ with effects (E_σ^+, E_σ^-) *covers* an environment transition $\tau = (s, a, s')$, denoted $a^\uparrow \models \tau$, if (1) the ground NSRT is applicable in s; (2) $E_\sigma^+ = \textsc{Abstract}(s') \setminus \textsc{Abstract}(s)$; and (3) $E_\sigma^- = \textsc{Abstract}(s) \setminus \textsc{Abstract}(s')$.

We assume that the following *weak semantics* connect P and E with the environment: for each ground NSRT $a^\uparrow$, there exists a state $s \in \mathcal{S}$ and there exists an action $a \in \mathcal{A}$ such that $a^\uparrow \models (s, a, f(s, a))$. Importantly, this means that the abstraction defined by the NSRTs does *not* satisfy downward refinability [8], which would have required the "there exists a state" to be "for all states." These weak semantics make learning efficient (Section VI), but require integrated planning (Section V).

To plan, it is important to be able to simulate the effects of actions on the continuous environment state. We next discuss the low-level transition model h, which is used for this.

Definition 5: Given a state s and ground NSRT $a^\uparrow$ with substitution σ, the *context of s for $a^\uparrow$* is $v_\sigma(s) = s[\sigma(o_1)] \circ \cdots \circ s[\sigma(o_k)]$, where $v_\sigma(s) \in \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$, $s[\cdot]$ looks up an object's attribute vector in s, and $\circ$ is vector concatenation.

In words, the context for a ground NSRT is the subset of a state's attribute vectors that correspond to the ground NSRT's objects, assembled into a vector. The context is the input to the low-level neural transition model h:

$$h(v_\sigma(s), a) \approx v_\sigma(f(s, a)),$$

where, recall, f is the *unknown* environment transition model. All objects not in σ are predicted to be unchanged.

Finally, the neural action sampler π of an NSRT connects the abstract and environment action spaces: it samples continuous actions from the environment action space $\mathcal{A}$ that lead to the NSRT's symbolic effects. Given a state s and applicable ground NSRT with substitution σ, if $a \sim \pi(\cdot \mid v_\sigma(s))$, then $(s, a, f(s, a))$ should ideally be covered by the ground NSRT. The fact that π is stochastic can be useful for planning, where multiple samples may be required to achieve desired effects (see Figure 1, or [6]).

There are three key properties of NSRTs to take away from these definitions. (1) NSRTs are fully relational, i.e., invariant over object identities. This leads to data-efficient learning and generalization to novel tasks and objects. (2) NSRTs do not assume downward refinability, as discussed above. (3) NSRTs are *locally scoped*; all components of a ground NSRT are defined only where it is applicable. This modularity leads to independent learning problems; see Section VI.

V. Neuro-Symbolic Planning with NSRTs

We now describe how NSRTs can be used to plan in a given task. Recall that the weak semantics of NSRTs (Section IV) do *not* guarantee downward refinability: a sequence of abstract actions that achieves a goal cannot necessarily be turned into a sequence of environment actions achieving that goal. Our strategy will be to perform integrated bilevel planning, with an outer search in the abstract space informing an inner loop producing environment actions. This planning strategy falls under the broad class of search-then-sample TAMP techniques [7]. *See Algorithm 1 for pseudocode.*

Symbolic Planning. We perform an outer A* search from $\textsc{Abstract}(s_0)$ to g, with the abstract transition model of Equation 1 and uniform action costs. For the search heuristic, we use h_{add}, a domain-independent heuristic from the symbolic planning literature [25] that approximates the state-to-goal distance under a delete relaxation of the abstract model. This A* search will find candidate *symbolic plans*: sequences of ground NSRTs $a^\uparrow \in \mathcal{A}^\uparrow$.

Continuous Planning. For each candidate symbolic plan, an inner loop attempts to refine it into a *plan* — a sequence of actions $a \in \mathcal{A}$ that achieves the goal g — using the neural components of the NSRTs. We use the action sampler π and low-level transition model h of each ground NSRT in the symbolic plan to construct an *imagined* state-action trajectory starting from the initial state s_0. If the goal g holds in the final imagined state, we are done. If g does not hold, or if any state's abstraction does not equal the expected abstract state according to the A* search, then we repeat this process. After n_{trials} (a hyperparameter) unsuccessful imagined trajectories, we return control to the A* search.

Algorithm Bilevel Planning with NSRTs
- **Input:** NSRT set $\{\langle O, P, E, h, \pi \rangle\}$
- **Input:** Task $\langle s_0, g, H \rangle$
- **Input:** n_{trials}: # of imagined trajectory tries
- `// A* with symbolic components of NSRTs and classical heuristics.`
- $s_0^\uparrow \leftarrow \textsc{Abstract}(s_0)$
- **for** $\overline{p} \in A^*(s_0^\uparrow, g, H, \{\langle O, P, E, \cdot, \cdot \rangle\})$ **do**
  - **for** n_{trials} *tries* **do**
    - Initialize `plan` as empty list
    - `// Imagine rollout with neural components of ground NSRTs.`
    - $s \leftarrow s_0$
    - **for** *ground NSRT* $\langle \cdot, \cdot, \cdot, \pi, h \rangle \in \overline{p}$ **do**
      - $a \sim \pi(\cdot \mid s)$ `// stochastic`
      - Append a to `plan`
      - $s \leftarrow h(s, a)$
    - **if** $g \subseteq \textsc{Abstract}(s)$ **then**
      - **return** `plan`

Algorithm 1: Pseudocode for bilevel planning with NSRTs. The outer loop runs A* search over the symbolic components of the NSRTs, from the symbolic initial state $s_0^\uparrow = \textsc{Abstract}(s_0)$ to the goal g. This A* produces candidate symbolic plans $\overline{p}$, which are sequences of ground NSRTs. The neural components of these ground NSRTs are used in the inner loop, which tries n_{trials} times to *refine* a symbolic plan into a sequence of continuous actions from the environment action space $\mathcal{A}$. If the goal g holds in the final state, we are done. In practice, we perform an extra optimization (not shown): we terminate the inner loop early whenever $\textsc{Abstract}(s)$ deviates from the expected sequence of states under $\overline{p}$.

Handling Failures. Recall that some transitions in the environment can lead to a failure state in $\mathcal{S}_{\text{fail}}$. Following [26], we would like to use the presence of a failure state during continuous planning to inform symbolic planning. We begin by assuming that we have a model which predicts whether a failure state is reached, and if so, the set of objects $\{o_1, \ldots, o_j\}$ that were responsible for the failure (e.g., two objects that were in collision, or an object that broke irreparably); we will show how to learn this model in Section VI. Now, we perform a domain-independent procedure: we introduce special predicates NotCausesFailure for every object type in the environment, and for each NSRT, we add a symbolic effect $\textsc{NotCausesFailure}(o_i)$ for each o_i in the parameters O. This says that every action affecting a set of objects absolves all those objects from being responsible for a failure; we found this simple technique sufficient for our experiments, but other, more domain-specific information can be leveraged instead [26]. During refinement, if a failure is predicted, we terminate the inner loop, update the preconditions of the ground NSRT at that timestep to include $\{\textsc{NotCausesFailure}(o_1), \ldots, \textsc{NotCausesFailure}(o_j)\}$ (where $\{o_1, \ldots, o_j\}$ are the set of objects predicted to be

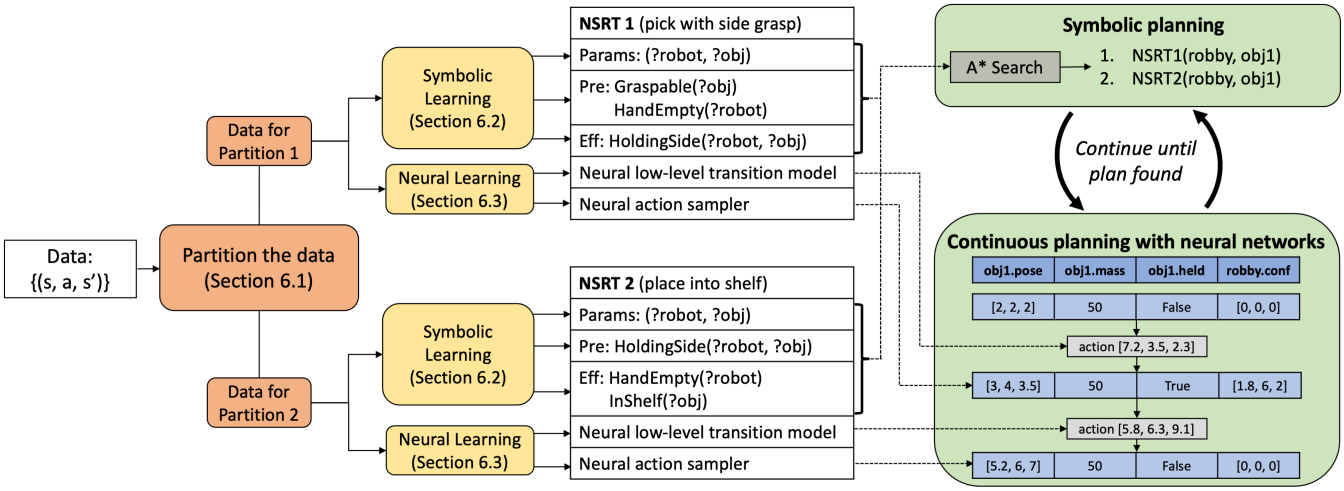

Fig. 2: Our pipeline, with a simplified Painting example. An NSRT (Section IV) contains both symbolic components used for A* search with AI planning heuristics, and neural components used for continuous planning. The example NSRTs shown in the middle require that a robot must be side-grasping an object to place it into a shelf. These NSRTs are *not* ground: their parameters are variables, so these NSRTs can be applied to any objects. We learn NSRTs from transition data (Section VI), and then use them to perform bilevel planning (Section V). Delete effects are omitted from this figure for visual clarity.

responsible for the failure), and restart A* from the initial state. This forces A* to either consider actions which change the states of these objects before using the same ground NSRT, or just avoid using this ground NSRT entirely.

VI. Learning NSRTs

We now address the problem of learning the structure (Section VI-A), the symbolic components (Section VI-B), and the neural components (Section VI-C) of NSRTs, all using the training dataset $\mathcal{D}$ (Section III).

A. Partitioning the Transition Data

Recall that $\mathcal{D}$ contains a set of samples from the unknown transition model f: each sample is a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$, and either a next state in $s' \in \mathcal{S}$ or a failure state in $\mathcal{S}_{\text{fail}}$. We will ignore the transitions that led to $\mathcal{S}_{\text{fail}}$; they will be used in Section VI-D. We begin by partitioning the set of transitions $\tau = (s, a, s')$ so that each partition $\psi \in \Psi$ will correspond to a single NSRT, thus automatically determining the number of learned NSRTs. Two transitions belong to the same partition iff their symbolic effects can be *unified*:

Definition 6: Two transitions τ_1 and τ_2 can be *unified* if there exists a bijective mapping σ from the objects in $\textsc{Eff}(\tau_1)$ to the objects in $\textsc{Eff}(\tau_2)$ s.t. $\sigma[\textsc{Eff}(\tau_1)] = \textsc{Eff}(\tau_2)$, where $\textsc{Eff}(\tau) = (\textsc{Abstract}(s') \setminus \textsc{Abstract}(s), \textsc{Abstract}(s) \setminus \textsc{Abstract}(s'))$, and $\sigma[\cdot]$ denotes substitution following σ.

These partitions can be computed in time linear in the number of transitions, objects, and atoms per effect set.

B. Learning the Symbolic Components

We now show how to learn NSRT parameters O, symbolic preconditions P, and symbolic effects E for each partition $\psi \in \Psi$. First, we define a mapping $\textsc{Ref}$ that maps a transition τ to a subset of objects in τ that are "involved" in the transition. In practice, we implement $\textsc{Ref}(\tau)$ by selecting all objects that appear in $\textsc{Eff}(\tau)$.[1] By construction of our partitions, every transition $\tau \in \psi$ will have equivalent $\textsc{Ref}(\tau)$, up to object renaming. We thus introduce NSRT parameters O corresponding to the types of all the objects in any arbitrarily chosen transition's $\textsc{Ref}(\tau)$. For each $\tau \in \psi$, let σ_τ be a bijective mapping from these parameters O to the objects in $\textsc{Ref}(\tau)$. The NSRT symbolic effects follow by construction: $E = \sigma_\tau^{-1}[\textsc{Eff}(\tau)]$ for any arbitrary $\tau \in \psi$.

To learn the symbolic preconditions P for the NSRT corresponding to partition ψ, we use a simple inductive approach that restricts learning by assuming that for each lifted effect set seen in the data, there is exactly one lifted precondition set.[2] By this assumption, the preconditions follow from an intersection of projected abstract states:

$$P = \bigcap_{\tau=(s,\cdot,\cdot)\in\psi} \sigma_\tau^{-1}[\textsc{Project}(\textsc{Abstract}(s))],$$

where $\textsc{Project}$ maps $\textsc{Abstract}(s)$ to the subset of atoms whose objects are all in $\textsc{Ref}(\tau)$. By construction, the semantics we defined in Section IV are satisfied over the training dataset: each transition belongs to one partition, and the preconditions for that partition must hold in its abstract state.

C. Learning the Neural Components

We now describe how to learn a low-level transition model h and action sampler π for each partition's NSRT. The key idea is to use the state projections computed during partitioning to create regression problems. Recalling Definition 5, let $v_\sigma = s[\sigma_\tau(o_1)] \circ \cdots \circ s[\sigma_\tau(o_k)]$ denote the context of state s from transition τ, where $(o_1, o_2, \ldots, o_k)$ are the NSRT parameters. In words, v_σ is a vector of the attribute values

[1] This suffices for our experiments, but it cannot capture "indirect effects," where some objects influence a transition without themselves changing; other implementations of $\textsc{Ref}$ could be used instead.

[2] See [24] for a more expensive method that avoids this assumption.

in state s corresponding to the objects that map the ground atoms $\textsc{Eff}(\tau)$ of the transition to the lifted effects E of the NSRT. We can do the same to produce $v_{\sigma'}$ for s'. Applying this to all transitions in ψ gives us a dataset of $(v_\sigma, a, v_{\sigma'})$.

Recall that we want to learn h such that $h(v_\sigma(s), a) \approx v_\sigma(f(s, a))$. With the dataset above, this learning problem now reduces to regression, with v_σ and a being the inputs and $v_{\sigma'}$ being the output. We use a fully connected neural network (FCN) as the regressor, trained to minimize mean-squared error. Learning π requires *distribution* regression, where we fit $P(a \mid v_\sigma)$ to the transitions $(v_\sigma, a, \cdot)$. We use an FCN that takes v_σ as input and predicts the mean μ and covariance matrix Σ of a Gaussian. This FCN is trained to maximize the likelihood of action a under $\mathcal{N}(\mu, \Sigma)$.[3] Since Gaussians have limited expressivity, we also learn an *applicability classifier* that maps pairs (v_σ, a) to 0 or 1, implemented as an FCN with binary cross-entropy loss. We implement π as a rejection sampler that draws from the Gaussian until the applicability classifier returns a 1.[4]

D. Learning to Predict Failures

Here we address the problem of learning to anticipate failures during planning. Note that unlike NSRT learning (Section VI), which is "locally scoped" to a fixed number of objects defined by the NSRT parameters, failure prediction can require reasoning about all objects in the full state. Extracting the training data transitions that led to a failure state in $\mathcal{S}_{\text{fail}}$, we create a dataset of the form $\{(s, a, \emptyset_{\text{fail}})\}$, where $\emptyset_{\text{fail}}$ is the set of objects in the failure state. We then train a graph neural network (GNN) that takes as input s, $\textsc{Abstract}(s)$, and a, and outputs a score between 0 and 1 for each object, representing the predicted probability that it is included in $\emptyset_{\text{fail}}$. We follow [27] for the graph encoding and GNN architecture. Once trained, we use the GNN to predict both whether a transition to a failure state occurs and $\emptyset_{\text{fail}}$, by checking whether there are any objects whose score is over 0.5, and including them in $\emptyset_{\text{fail}}$ if so.

VII. Experiments

Our empirical evaluations address the following key questions: **(Q1)** Can NSRTs be learned data-efficiently? **(Q2)** Can learned NSRTs be used to plan to long horizons, especially in tasks involving new and more objects than were seen in the training dataset? **(Q3)** Is bilevel planning efficient and effective, and are both levels needed? **(Q4)** To what extent are learned action samplers useful for planning?

A. Experimental Setup

We evaluate Q1-Q4 by running seven methods on four environments. All experiments were run on Ubuntu 18.04 using 4 CPU cores of an Intel Xeon Platinum 8260 processor.

Environments. In this section, we describe our four environments. The environments are illustrated in Figure 1 (bottom row). Each environment has two sets of tasks: "easy" test and "hard" test. "Hard" test tasks require generalization to more objects. In all environments, we transition to a failure state in $\mathcal{S}_{\text{fail}}$ whenever a geometric collision occurs.

- *Environment 1:* In "PickPlace1D," a robot must pick blocks and place them into designated target regions on a table. All poses are 1D. Some placements are obstructed by movable objects; none of the predicates capture obstructions, causing a lack of downward refinability.
- *Environment 2:* In "Kitchen," a robot waiter in 3D must pick cups, fill them with water, wine, or coffee, and serve them to customers. Some cups are too heavy to be lifted; the cup masses are not represented by the predicates, causing a lack of downward refinability.
- *Environment 3:* In "Blocks," a robot in 3D must stack blocks on a table to make towers. In this environment only, the downward refinability assumption holds.
- *Environment 4:* In "Painting," a robot in 3D must pick, wash, dry, paint, and place widgets into a box or shelf. Placing into the box (resp. shelf) requires picking with a top (resp. side) grasp. All widgets must be painted a particular color before being placed, which first requires washing/drying if the widget starts off dirty or wet. The box has a lid that may obstruct placements; whether the lid will obstruct a placement is not represented symbolically, causing a lack of downward refinability.

Dataset Creation. We create the training dataset $\mathcal{D}$ for each environment by, 700 times, sampling an initial state s_0 and running a scripted stochastic policy π_0 from it. This policy avoids the zero-measure issues that would arise from uniformly random actions, but it is not goal-directed, and it does not nearly suffice to solve test tasks. For PickPlace1D, Kitchen, and Blocks, we experiment with up to 7000 transitions; for Painting, up to 12000 since it is more challenging.

Methods Evaluated. We evaluate the following methods. *Note that B4-B6 receive information that Ours and B1-B3 do not have access to: the scripted stochastic policy π_0 mentioned above.* All methods get the same training dataset.

- *Ours: Bilevel planning with NSRTs.* This is our main approach. Plans are executed open-loop.
- *B1: Symbolic planning only.* This baseline performs symbolic planning using the symbolic components of the learned NSRTs. When a symbolic plan is found that reaches the goal, it is immediately executed by calling the learned action samplers for the corresponding ground NSRTs in sequence, open-loop. The low-level transition models are not used. This baseline ablates away our integrated planner and assumes downward refinability.
- *B2: Neural planning only with forward shooting.* This baseline randomly samples H-length sequences of ground NSRTs and uses their neural components to imagine a trajectory, repeating until it finds a trajectory where the final state satisfies the goal. This baseline does not use the symbolic components of the NSRTs, and thus can be seen as an ablation of the symbolic planning.
- *B3: Neural planning only with hill climbing.* This baseline performs local search over full plans. At each

[3]Here, we are assuming that the desired action distribution has nonzero measure. In practice, Σ can be arbitrarily small.

[4]If the applicability classifier fails enough times (10 in experiments), we terminate the inner loop and continue the outer A* search (see Alg. 1).

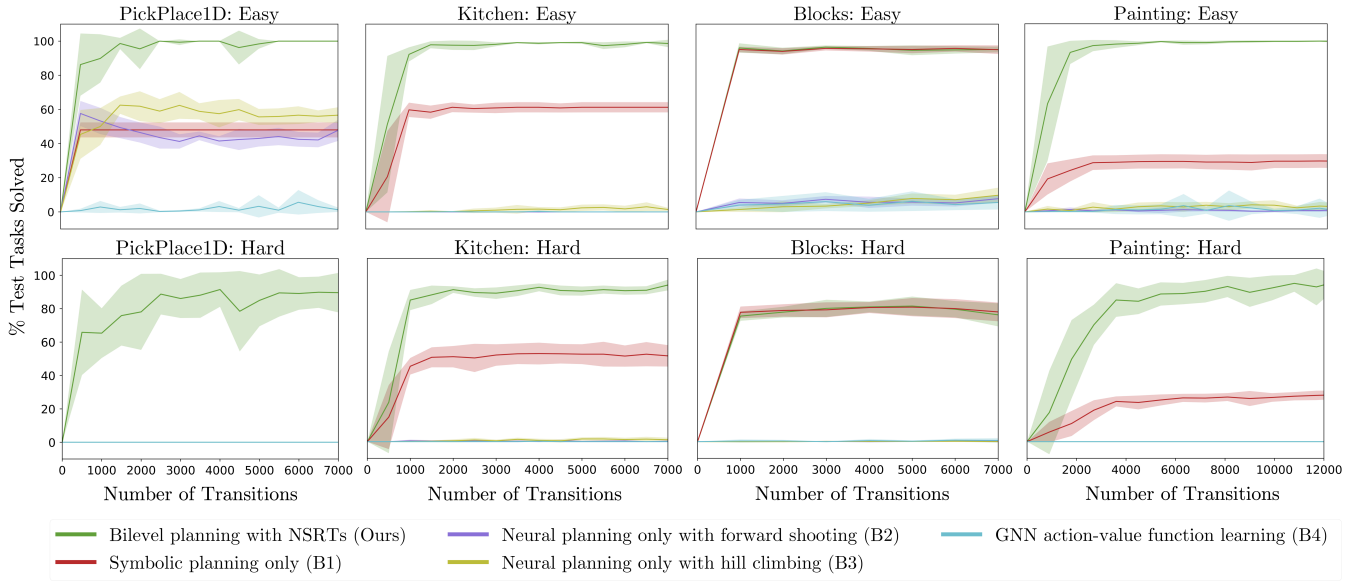

Fig. 3: Learning curves showing the percentage of 100 randomly generated test tasks (top row: easy tasks; bottom row: hard tasks) solved versus the number of transitions in the dataset. Each curve depicts a mean over 8 seeds, with standard deviation shaded. All methods have a timeout of 3 seconds per task. NSRTs (green) quickly learn to solve many more tasks than the baselines, especially in the hard tasks.

iteration, a random plan step is resampled using the learned action sampler of a random NSRT. The new plan is rejected unless it improves the number of goal atoms satisfied in the final imagined state. As in B2, the symbolic components of the NSRTs are not used.

- *B4: GNN action-value function learning.* This "model-free" baseline trains a goal-conditioned graph neural network (GNN) action-value function using fitted Q-iteration. The GNN takes as input a continuous low-level state, the corresponding abstract state, and a continuous action; it outputs a value. At evaluation time, given a state, we draw candidate actions from π_0 (see above).
- *B5: No learned samplers.* This baseline is an ablation of our main approach that does not use the learned NSRT action samplers π. Instead, actions are drawn from π_0.
- *B6: No symbolic components or learned samplers.* This baseline is an ablation that uses the forward shooting of B2 but with actions drawn from π_0, like B5. Only the low-level transition models h are used.

Additional details. All neural networks are fully connected with two hidden layers of size 32, and trained using the Adam optimizer for 35K (action samplers), 10K (low-level transition models), or 50K (applicability classifier) epochs with a learning rate of 1e-3. In our robotic environments of interest, transitions are often *sparse*, changing only a subset of object attributes at any given time. For learning the low-level transition model, we exploit this by calculating the attributes that change in *any* transition within a partition, and only predict next values for those attributes. For learning the action samplers, we restrict the covariance matrix Σ to be diagonal and positive semi-definite using an exponential linear unit [28]. During evaluation, we clip samples from the action samplers to be at most 1 standard deviation from the mean, for improved stability. The applicability classifier is trained with negative examples collected from either data in other partitions, or data in the same partition but with the objects re-mapped. We subsample negative examples to ensure that the dataset is balanced in a 1:1 ratio with the positive examples. In all experiments, we use $n_{\text{trials}} = 1$, which we found to be sufficient due to the accuracy of the action samplers and low-level transition models. For the action-value function (B4), we train by running 5 iterations of fitted Q-iteration, and during evaluation, we sample 100 candidate actions from the scripted policy π_0 at each step, choosing the action with the best predicted value to execute in the environment. Methods that use shooting (B2 and B6) try up to 1000 iterations, or until the timeout (3 seconds for every method across all experiments) is reached.

B. Results and Discussion

See Figure 3 for learning curves. The main observation is that in all environments, our method quickly learns to solve tasks within the allotted 3-second timeout. Thus, **Q1** and **Q2** can be answered affirmatively. Turning to **Q3**, we can study whether bilevel planning is effective by comparing Ours, B1, and B2. The gap between Ours and B1 shows the importance of integrated bilevel planning. B1 will not be effective in any environment where downward refinability does not hold; only Blocks is downward refinable, which explains the identical performance of Ours and B1 there. B2 fails in most cases, confirming the usefulness of the symbolic components of the learned NSRTs.

Both B3 and B4 are generally ineffective. B3 performs local search, which is much weaker than our directed A*. B4 is model-free, forgoing planning in favor of learning a value function; such strategies are known to be more data-hungry [4]. In our experimentation, we found that for the

	PickPlace1D		Kitchen		Blocks		Painting	
Methods	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Bilevel planning with NSRTs (Ours)	**98.4**	**85.0**	**99.1**	**90.4**	**95.0**	**79.6**	**99.6**	**89.6**
No learned samplers (B5)	**95.9**	46.4	71.9	32.6	89.9	53.4	84.5	0.1
No symbolic components or learned samplers (B6)	71.1	0.0	0.0	1.5	62.9	8.6	5.4	0.0

TABLE I: Percentage of 100 randomly generated test tasks solved after learning on the full training dataset. Each number is a mean over 8 seeds; bold results are within one standard deviation of best. Both the symbolic components *and* the learned samplers are critical.

Easy test tasks, B4 starts performing decently after seeing about four times as much data as we used in making the plots, confirming that it requires substantially more data.

To evaluate **Q4**, we turn to an ablation study. Table I compares our method with B5 and B6, both of which sample actions from the scripted policy π_0 rather than using our learned NSRT action samplers. First, comparing B5 and B6, bilevel planning is much better than shooting, which speaks to the benefits of using the symbolic components of the NSRTs to guide planning; this conclusion was also supported by Figure 3. Second, comparing Ours and B5, the learned action samplers help substantially. This is because π_0 is highly generic, not targeted toward any specific set of effects like our NSRT action samplers are.

VIII. Conclusion, Limitations, and Future Work

We proposed NSRTs for long-horizon, goal-based, object-oriented planning tasks, showed that their neuro-symbolic structure affords fast bilevel planning, and found that they are data-efficient to learn and effective at generalization, outperforming several baselines. Key limitations of this work include the assumption that predicates are given and the assumption that environments are deterministic and fully observable. To address the former, NSRTs could be combined with work on learning predicates from high-dimensional inputs [16]. For the latter, we hope to draw on TAMP techniques for stochastic and partially observed settings.

References

[1] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," *arXiv preprint arXiv:1712.05474*, 2017.

[2] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning," *GitHub*, 2016.

[3] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[4] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv:2006.16712*, 2020.

[5] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2970–2977.

[6] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 866–894, 2021.

[7] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[8] B. Marthi, S. J. Russell, and J. A. Wolfe, "Angelic semantics for high-level actions." in *ICAPS*, 2007, pp. 232–239.

[9] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," *Machine learning*, vol. 43, no. 1, pp. 7–52, 2001.

[10] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems*, 2016.

[11] V. Xia, Z. Wang, K. Allen, T. Silver, and L. P. Kaelbling, "Learning sparse relational transition models," in *International Conference on Learning Representations*, 2019.

[12] T. Lang, M. Toussaint, and K. Kersting, "Exploration in relational domains for model-based reinforcement learning," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3725–3768, 2012.

[13] A. Dittadi, F. K. Drachmann, and T. Bolander, "Planning from pixels in atari with learned symbolic representations," *arXiv preprint arXiv:2012.09126*, 2020.

[14] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[15] A. Arora, H. Fiorino, D. Pellier, M. Etivier, and S. Pesty, "A review of learning planning action models," *Knowledge Engineering Review*, vol. 33, 2018.

[16] M. Asai, "Unsupervised grounding of plannable first-order logic representation from images," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 583–591.

[17] A. Ahmetoglu, M. Y. Seker, A. Sayin, S. Bugur, J. Piater, E. Oztop, and E. Ugur, "Deepsym: Deep symbol generation and rule learning from unsupervised continuous robot interaction for planning," *arXiv preprint arXiv:2012.02532*, 2020.

[18] J. Hoffmann, "FF: The fast-forward planning system," *AI magazine*, vol. 22, no. 3, pp. 57–57, 2001.

[19] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 540–550.

[20] H. Kokel, A. Manoharan, S. Natarajan, R. Balaraman, and P. Tadepalli, "RePReL: Integrating relational planning and reinforcement learning for effective abstraction," in *Thirty First International Conference on Automated Planning and Scheduling (ICAPS)*, 2021.

[21] P. Bercher, R. Alford, and D. Höller, "A survey on hierarchical planning-one abstract idea, many concrete realizations." in *IJCAI*, 2019, pp. 6267–6275.

[22] H. H. Zhuo, D. H. Hu, C. Hogg, Q. Yang, and H. Munoz-Avila, "Learning htn method preconditions and action models from partial observations," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[23] J. Loula, K. Allen, T. Silver, and J. Tenenbaum, "Learning constraint-based planning models from demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[24] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Perez, "Learning symbolic operators for task and motion planning," *arXiv preprint arXiv:2103.00589*, 2021.

[25] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 5–33, 2001.

[26] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 639–646.

[27] T. Silver, R. Chitnis, A. Curtis, J. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[28] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations*, 2016.