Toward Compositional Generalization in Object-Oriented World Modeling

Linfeng Zhao [1] **Lingzhi Kong** [1] **Robin Walters** [1] **Lawson L.S. Wong** [1]

Abstract

Compositional generalization is a critical ability in learning and decision-making. We focus on the setting of reinforcement learning in *object-oriented environments* to study compositional generalization in world modeling. We (1) formalize the compositional generalization problem with an *algebraic* approach and (2) study how a world model can achieve that. We introduce a conceptual environment, Object Library, and two instances, and deploy a principled pipeline to measure the generalization ability. Motivated by the formulation, we analyze several methods with *exact* or *no* compositional generalization ability using our framework, and design a differentiable approach, Homomorphic Object-oriented World Model (HOWM), that achieves soft but more efficient compositional generalization.

1. Introduction

In learning and decision-making, the goal is to train models and agents that generalize to new data, tasks, and environments. Recently, there has been significant interest in learning transition models for object-based environments in computer vision and reinforcement learning, in particular from images (Burgess et al., 2019; Watters et al., 2019; Kipf et al., 2019; Kossen et al., 2019; Lin et al., 2020; Veerapaneni et al., 2020; Locatello et al., 2020). These environments are naturally factorized by their objects.

In this paper, we aim to study one widely existing form of generalization that has not been formally studied in object-oriented world modeling: *compositional generalization*. That is, if we have seen various combinations of objects during training, but are presented with a novel combination at evaluation, we should still expect our agents to recognize familiar objects seen during training and predict their effects appropriately. While compositional generalization has been previously studied in natural language (Johnson et al., 2017; Lake & Baroni, 2018; Bahdanau et al., 2019; Gordon et al., 2019; Keysers et al., 2020), such work does not yet exist for (action-conditioned) world modeling.

To study compositional generalization, we design a class of object-oriented environments called Object Library, illustrated in Figure 1 left. Such environments feature K objects, drawn from a library of N objects; the K objects remain the same during each episode. Although visually distinct, these environments are isomorphic to each other (Figure 1 center). Furthermore, the isomorphism is factorized by the constituent objects, so we should expect our models to generalize to new scenes (combinations of objects), as long as the constituent objects have been observed in some scenes during training. For example, if during training we have observed scenes containing {▲, ▼} and {◀, ▶}, then our models should also make accurate predictions in novel scenes {▲, ◀} and {▼, ▶}.

We formally define compositional generalization as the ability to generalize from a scene (with a combination of object) to another scene with *replaced* objects, or *equivariance* to *object-replacement operation*. This bridges the behavioral perspective (generalizing to replaced object) with functional implementation (equivariant model). The proposed framework illustrates two paths for compositional generalization in world modeling: (1) *exact* compositional generalization with potentially more intensive resource usage, (2) learning *soft* object and action binding in *latent* space. Specifically, for the latter path, one primary challenge comes from the lack of canonical ordering of objects in images. We prove that, if it learns to correctly *bind actions* with *latent object slots*, this learned path can still achieve *perfect* compositional generalization, with far less resources needed.

Based on this analysis, we propose an soft approach for learning compositional generalizable world models, called Homomorphic Object-oriented World Model (HOWM), which deploys an *Action Attention* module to bind actions and can be trained differentiably with the *Aligned Loss*. We analyze the compositional generalization ability of existing methods and HOWM using instances of Object Library, and demonstrate that HOWM generalizes well with fewer resources. The idea of learning action bind-

[1]Khoury College of Computer Sciences, Northeastern University, MA. Correspondence to: Linfeng Zhao <zhao.linf@northeastern.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

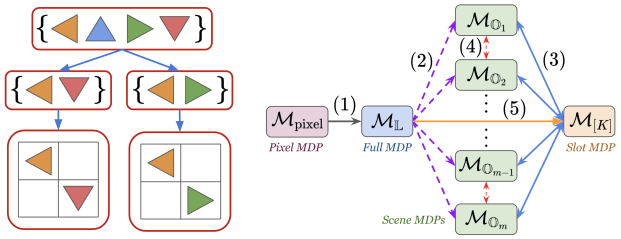

Figure 1: **(Left)** An example of our Object Library environment: Rush Hour, described in Section 4.1. Each scene features $K = 2$ objects from a library of $N = 4$ objects. **(Right)** A family of MDPs for modeling Object Library, as explained in Section 3.3.

ing for compositional generalization is generic and can be plugged into world models for more complex environments or object-based planning. More resources are available under `http://lfzhao.com/oowm`.

2. Background

Our environments are modeled as Markov decision processes. A Markov decision process (MDP) is a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$.

MDP homomorphisms. In this paper, we will study families of related MDPs using the framework of MDP homomorphisms (Ravindran & Barto, 2004; van der Pol et al., 2020). An *MDP homomorphism* $h\colon \mathcal{M} \to \overline{\mathcal{M}}$ is a mapping from one MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ to another $\overline{\mathcal{M}} = \langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{T}, \overline{R}, \gamma \rangle$. The map h consists of a tuple of surjective maps $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$, where $\phi\colon \mathcal{S} \to \overline{\mathcal{S}}$ is the state mapping and $\alpha_s\colon \mathcal{A} \to \overline{\mathcal{A}}$ is the *state-dependent* action mapping. The mappings are constructed to satisfy the following conditions for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$\begin{aligned} \overline{R}\left(\phi(s), \alpha_s(a)\right) &= R(s, a), \\ \overline{T}\left(\phi\left(s'\right) \mid \phi(s), \alpha_s(a)\right) &= \sum_{s'' \in \phi^{-1}(\phi(s'))} T\left(s'' \mid s, a\right). \end{aligned} \tag{1}$$

We call the *reduced* MDP $\overline{\mathcal{M}}$ the *homomorphic image* of $\mathcal{M}$ under h. If $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$ has *bijective* maps ϕ and $\{\alpha_s\}$, we call h an *MDP isomorphism*.

Symmetry groups and equivarance. In Section 4, we will formalize compositional generalization in object-oriented environments as a type of "object-replacement" symmetry. In mathematics, a *symmetry group* G is an algebraic concept, denoting the collection of all symmetric transformations of an entity, satisfying the axioms of associativity, identity, inverse, and closure. A (left) *group operation* (action) of a group G on a set $\mathcal{X}$ is defined as the mapping $(g, x) \mapsto g.x$. If f is a function $f : \mathcal{X} \to \mathcal{Y}$ and G acts on $\mathcal{X}$ and $\mathcal{Y}$, then f is an *equivariant map* if it commutes with the operation of the group: $g.f(x) = f(g.x), \forall g \in G, \forall x \in \mathcal{X}$. The function f is considered G-*equivariant*. If instead $f(x) = f(g.x)$ holds, then f is considered G-*invariant*.

MDP homomorphisms with symmetry. The above concepts can be connected together. Given group G, an MDP homomorphism h is said to be *group structured* if any state-action pair (s, a) and its transformed counterpart $g.(s, a)$ are mapped to the same abstract state-action pair: $(\phi(s), \alpha_s(a)) = (\phi(g.s), \alpha_{g.s}(g.a))$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$. For convenience, we denote $g.(s, a)$ as $(g.s, g.a)$, where $g.a$ implicitly[1] depends on state s. Applied to the transition and reward functions, the transition function T is G-equivariant if T satisfies $T(g.s'|g.s, g.a) = T(s'|s, a)$, and reward function R is G-invariant if $R(g.s, g.a) = R(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$.

This section is intentionally brief; see van der Pol et al. (2020) and Ravindran & Barto (2004) for a more in-depth account of MDPs with symmetries. To keep the exposition simple in the main text, we focus on predicting transition dynamics only and assume that transitions are deterministic. The derivations extend naturally to stochastic environments and rewards (Appendix F and E.1).

3. Object-Oriented Environments

In this section, we define the concept of object-oriented environments (OOE), and introduce a specific family of OOEs, Object Library, for studying composition generalization. We model these environments as MDPs, and in the process we will define multiple related MDPs for Object Library.

3.1. Image-based environments and factorization

We study image-based environments with multiple objects. At each time step, the agent observes an image $s_t \in \mathcal{S}$, consisting of multiple objects that fully describe the current *scene*, and takes an action $a_t \in \mathcal{A}$ to control a single object. We model this as an MDP with image-based states, $\mathcal{M}_{\text{pixel}}$. What distinguishes $\mathcal{M}_{\text{pixel}}$ from an arbitrary high-dimensional MDP with a high-dimensional state space is that the environment actually has low-dimensional factorized structure, in that the environment is fully determined solely by the objects in the scene and their relations (Ravindran & Barto, 2004; Diuk et al., 2008).

[1]The group operation acting on action space $\mathcal{A}$ *depends on state*, since G actually acts on the *product space* $\mathcal{S} \times \mathcal{A}$, $(g, (s, a)) \mapsto g.(s, a)$, while we denote it as $(g.s, g.a)$ for consistency with $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$. As a bibliographical note, in van der Pol et al. (2020), the group acting on state and action space is denoted as state transformation $L_g\colon \mathcal{S} \to \mathcal{S}$ and *state-dependent* action transformation $K_g^s\colon \mathcal{A} \to \mathcal{A}$.

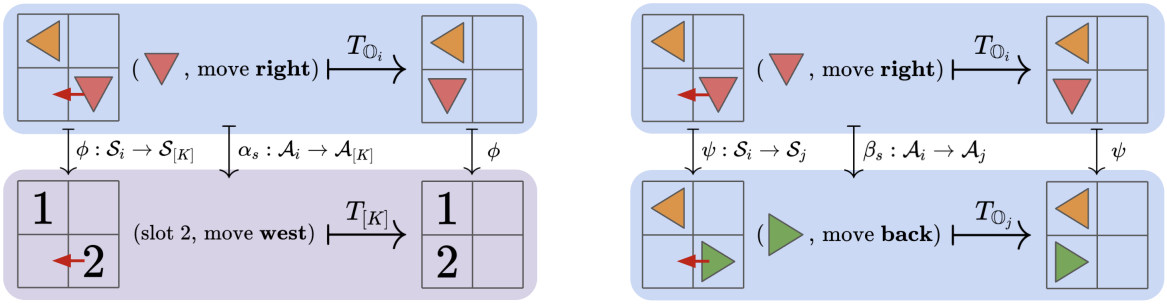

Figure 2: An illustrative commutative diagram. **(Left)** For any scene MDP $\mathcal{M}_{\mathbb{O}_i}$, we can bind its objects and actions to the representative slot MDP $\mathcal{M}_{[K]}$. Note that the *object-relative* action `"right"` needs to be converted to the *absolute representative* action `"west"` in $\mathcal{M}_{[K]}$. **(Right)** Since the binding between objects (and their actions) and slots is one-to-one, we can relate two scene MDPs by composing the binding $h_{i\to j} = h_i \circ h_j^{-1}$; the two MDPs are isomorphic.

More formally, we define $\mathcal{M}_{\text{pixel}}$ to be an object-oriented environment (OOE) if there exists an MDP homomorphism $\mathcal{M}_{\text{pixel}} \to \overline{\mathcal{M}}$ that maps the image-based MDP to a factored MDP $\overline{\mathcal{M}}$ (Guestrin et al., 2003). $\overline{\mathcal{M}}$ has factorized state and action spaces $\overline{\mathcal{S}} = \overline{\mathcal{S}}_1 \times \ldots \times \overline{\mathcal{S}}_N, \overline{\mathcal{A}} = \overline{\mathcal{A}}_1 \times \ldots \times \overline{\mathcal{A}}_N$, where each factor is meant to model one of N objects, and ideally has significantly lower dimension. We explain why we choose factorized action space in Appendix B.2.

3.2. Object Library

In this paper, we consider compositional generalization within a family of OOEs, which we call *Object Library*. Object Library is an OOE augmented with a library of all possible objects $\mathbb{L} = \{o_1, \ldots, o_N\}$, of which only K objects are present in any given scene, where $1 < K < N$. Note that the library is not given and is meant to be learned from images; we merely introduce it as a conceptual tool.

In each episode, the K objects are persistent, but between episodes a different set of K objects may be chosen from $\mathbb{L}$. This allows us to generate different scenes with different object sets during training and evaluation, thus enabling us to measure the performance discrepancy when faced with different *object compositions*. Compositional generalization will be formally defined in the next section.

3.3. Modeling Object Library as a family of MDPs

We now define several MDPs for modeling Object Library, illustrated in Figure 1.

- Pixel MDP $\mathcal{M}_{\text{pixel}}$: This is the original environment with image-based states.
- Full MDP $\mathcal{M}_{\mathbb{L}}$: We define $\mathcal{M}_{\mathbb{L}}$ to be the latent factored MDP, with one state/action factor per object: $\mathcal{S}_{\mathbb{L}} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_N$, and likewise for $\mathcal{A}_{\mathbb{L}} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$. Since there are only K objects in the scene, each $\mathcal{S}_i$ contains a special null state ε that indicates the object is not present.
- Scene MDP $\mathcal{M}_{\mathbb{O}}$: Since the K objects persist in any particular instance of the Object Library environment, both $\mathcal{M}_{\text{pixel}}$ and $\mathcal{M}_{\mathbb{L}}$ are partitioned into $\binom{N}{K}$ connected components, or *sub-MDPs* (Ravindran & Barto, 2004). It is convenient to consider each of these sub-MDPs, which we call the *scene MDP* $\mathcal{M}_{\mathbb{O}}$. More formally, let $\mathbb{O} = \{i_1, \ldots, i_K\}$ be the indices of the K present objects. We construct $\mathcal{M}_{\mathbb{O}}$ by projecting $\mathcal{S}_{\mathbb{L}}$ and $\mathcal{A}_{\mathbb{L}}$ to $\mathbb{O}$: $\mathcal{S}_{\mathbb{O}} = \mathcal{S}_{i_1} \times \ldots \times \mathcal{S}_{i_K}$, and likewise for $\mathcal{A}_{\mathbb{O}}$.
- Slot MDP $\mathcal{M}_{[K]}$: We will eventually show that all scene MDPs are isomorphic to each other. Key to this construction is the existence of a *representative* factored MDP among all scenes, which we denote as the *slot MDP* $\mathcal{M}_{[K]}$. $\mathcal{M}_{[K]}$ has K object "slots": $\mathcal{S}_{[K]} = \mathcal{S}_1 \times \ldots \mathcal{S}_K$, and likewise for $\mathcal{A}_{[K]}$. Ideally, we can "bind" every scene MDP $\mathcal{M}_{\mathbb{O}}$ to $\mathcal{M}_{[K]}$ *without* any loss of information, since a scene MDP has K objects and the slot MDP can represent the K objects using its K slots.

The relationship between these MDPs are shown in Figure 1 right. There exists an MDP homomorphism from the high-dimensional $\mathcal{M}_{\text{pixel}}$ to the low-dimensional factored $\mathcal{M}_{\mathbb{L}}$ (black arrow). Depending on the K objects present in the scene, $\mathcal{M}_{\mathbb{L}}$ can be projected onto one of $\binom{N}{K}$ scene MDPs $\mathcal{M}_{\mathbb{O}_i}$ (purple dashed arrow). Each of these $\mathcal{M}_{\mathbb{O}_i}$ is isomorphic to $\mathcal{M}_{[K]}$ (blue arrow, double headed), and therefore are also isomorphic to each other (red dashed arrow, double headed). Our analysis in Section 4 also assumes the existence of an MDP homomorphism from $\mathcal{M}_{\mathbb{L}}$ to $\mathcal{M}_{[K]}$ (orange arrow).

4. Compositional Generalization in OOEs

In OOEs, at least two types of generalization exist: (1) generalizing to *unseen objects*, and (2) generalizing to *unseen combinations* (scenes) of *known objects*. We consider type (2) as *compositional generalization*. This is similar to recent work on compositional generalization in natural

language (Gordon et al., 2019), where a sentence is viewed as an ordered set of words. In the context of Object Library, the object library $\mathbb{L}$ is our "vocabulary", and selecting novel sets of objects $\mathbb{O}$ to form scenes is similar to composing unseen sentences from novel combinations of words.

We first illustrate this key idea using a simple instance of Object Library. Next, we formally define our notion of compositional generalization. Finally, we provide a set of conditions that we prove are sufficient to achieve compositional generalization in the Object Library family of OOEs.

4.1. An illustrative example

We use an instance of Object Library, Rush Hour, to illustrate our ideas (see Figures 1 and 2). Rush Hour is motivated by a grid game, where multiple cars are in different absolute orientations: `(north, south, east, west)`. Each car has an action space `(forward, backward, left, right)` that moves it relative to its orientation. For example, if a car is facing `south`, moving `right` means moving `west` in the world. Consider an instance with $N = 4$ possible objects in the library $\mathbb{L}$ described by shape and orientation: {▲, ▼, ◀, ▶}. Each scene consists of $K = 2$ objects, so there are 6 possible scenes: { {▲, ▼}, {▲, ◀}, {▲, ▶}, {▼, ◀}, {▼, ▶}, {◀, ▶} }.

Object-replacement symmetry. We formalize the notion of mapping between scenes (with different object combinations) using permutation symmetry. The symmetry group $\Sigma_N = \Sigma_4$ acts naturally on $\mathbb{L}$ with 4 cars, by replacing a car with another; it induces an operation acting on scenes. For example, if we have a permutation mapping $\sigma(▼) = ▶, \sigma \in \Sigma_4$, and other cars remain the same, the induced operation on the scene would be $\sigma(\{◀, ▼\}) = \{◀, ▶\}$. If our learned transition model is equivariant to Σ_N, then we can generalize to novel scenes (e.g., Figure 2 bottom right) by appropriate permuting (replacing objects) from training scenes (Figure 2 top right).

Lifting from the slot MDP. We can observe that scenes with K different present objects are structurally similar to each other, motivating us to not consider all N objects independently, but to share knowledge between scenes. Furthermore, since all K-object scenes are similar, we could find a *representative* "scene" where all scenes are similar to it, and convert actions to consistent meaning (from relative `up` to absolute `north`, Figure 2 bottom left). We call it the *slot MDP* $\mathcal{M}_{[K]}$ (Figure 2 bottom left), where all scene MDPs are called *isomorphic* to it. In other words, by learning a world model in the slot MDP, we should automatically get good model for all scenes, or "lifting", and thus generalize.

Measuring compositional generalization. If we train on scene MDPs $\{\mathcal{M}_{\{▲,▼\}}, \mathcal{M}_{\{◀,▶\}}\}$, the model should ideally generalize to $\{\mathcal{M}_{\{▲,◀\}}, \mathcal{M}_{\{▼,▶\}}\}$. By training on the former and evaluating transition-function prediction errors on the latter, we can measure generalization error. We formalize this by defining the notion of *equivariance error* in the next section. See Section C for details.

4.2. Defining exact compositional generalization

Our definition is based on equivariance to object replacement in OOEs. Specifically, if the (learned) transition model $\hat{T}_{\mathbb{L}}$ for $\mathcal{M}_{\mathbb{L}}$ is equivariant to any object-set replacement, then $\hat{T}_{\mathbb{L}}$ can generalize to all object sets $\mathbb{O}$, including unseen combinations, as long as all individual objects in the library $\mathbb{L}$ have been observed in training scenes. This equivariance definition connects the behavioral perspective (how a system with compositional generalization should behave) and functional perspective (achieving by permutation equivariant networks). More formally, we define the object-replacement operation as follows:

Definition 4.1 (Object-replacement operation $p_{\mathbb{L}}$)**.** Let Σ_N denote the permutations of N elements, which acts naturally on the object-library set $\mathbb{L}$. The object-replacement operation is the group operation $\rho_{\mathbb{L}} \colon \Sigma_N \times \mathbb{L} \to \mathbb{L}$ where Σ_N acts on the indices in $\mathbb{L}$ to replace the identity of objects.

For example, using disjoint cycle notation, an object replacement operation $\pi = (123)(45)$ acts on $\mathbb{L}$ by sending $\pi(3) = 1$ and $\pi(4) = 5$. This induces a group operation on scenes, p_K (see Definition E.3 in Appendix E.2), that maps a set of objects to another set, e.g., $p_K(\pi, \{3, 4\}) = \{1, 5\}$. Thus, Σ_N permutes the scene MDPs $\{\mathcal{M}_{\mathbb{O}}\}$. We use $p_{\mathbb{L}}$ to define the measure of compositional generalization:

Definition 4.2 (Equivariance error of $\hat{T}_{\mathbb{L}}$ on $\mathcal{M}_{\mathbb{L}}$)**.** Let $\hat{T}_{\mathbb{L}} : \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}} \times \mathcal{S}_{\mathbb{L}} \to \mathbb{R}_+$ be a (learned) transition model of $\mathcal{M}_{\mathbb{L}}$. The sample equivariance error of $T_{\mathbb{L}}$ at $(s, a, s') \in \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}} \times \mathcal{S}_{\mathbb{L}}$ with respect to $\sigma \in \Sigma_N$ is defined

$$\lambda_{\mathbb{L}}^{\sigma} \triangleq \left| \hat{T}_{\mathbb{L}}(s' \mid s, a) - \hat{T}_{\mathbb{L}}(\sigma.s' \mid \sigma.s, \sigma.a) \right|. \qquad (2)$$

The equivariance error of $T_{\mathbb{L}}$ is then defined as the expectation $\lambda_{\mathbb{L}} = \mathbb{E}_{s,a,s',\sigma}[\lambda_{\mathbb{L}}^{\sigma}]$.

The magnitude of λ measures the failure of $\hat{T}_{\mathbb{L}}$ to be Σ_N-equivariant, hence the name *equivariance error*. For a perfectly Σ_N-equivariant transition function $T_{\mathbb{L}}$, the error is guaranteed to be 0 by the definition of equivariance. Note that for clarity, we only consider *transition* prediction in the above definition. A complete version of homomorphism also preserves the reward structure, thus preserving optimal values and policies. We provide the full version with rewards in Appendix E.1.

4.3. Achieving soft compositional generalization

Ideally, we can achieve compositional generalization according to the above metric by simply learning a Σ_N-equivariant

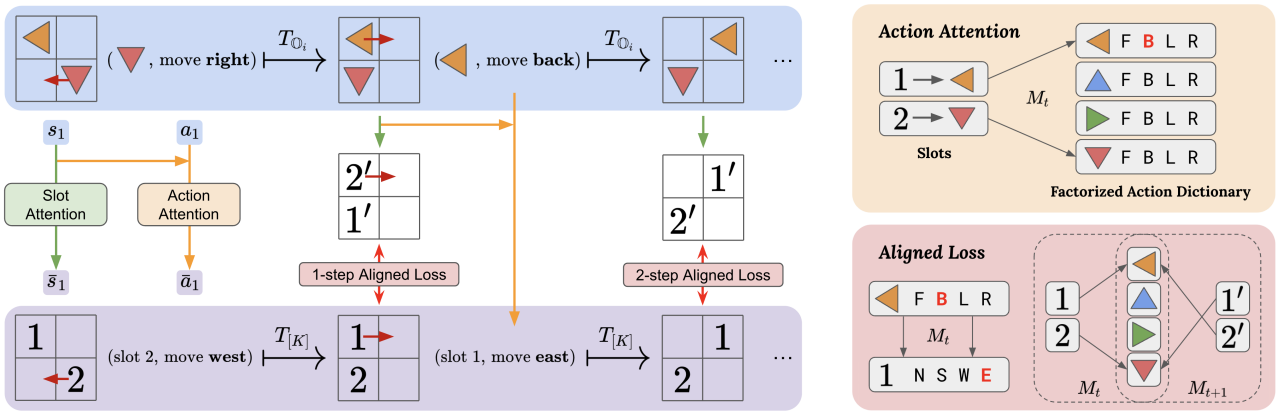

Figure 3: **(Left)** Overview of the world model prediction: the upper blue sequence is a ground pixel MDP with some objects $\mathbb{O}_i$, and the lower one is the slot MDP. We emphasize two facts: (1) encoded object slots in different steps may have different ordering (marked as $1'$ and $2'$), and (2) the transition model is equivariant in slot ordering, i.e., consistent across time steps (in 1 and 2), thus the loss computation needs alignment of slots (between $1, 2$ and $1', 2'$). **(Top right)** Action Attention learns to bind actions from interaction (action-object correspondence is *unknown* and learned). **(Bottom right)** In the Aligned Loss, the learned binding matrices M_t and M_{t+1} are used to lift slots in t and $t+1$ to a canonical space (full MDP).

transition model and correct binding of objects *and* actions. However, this is difficult in practice for large N (size of object library), as we show in our experiments. In this section, we derive an alternative path to Σ_N-equivariance that only requires us to learn a Σ_K-equivariant model, which is significantly more tractable since we expect $K \ll N$.

Instead of directly learning a Σ_N-equivariant model, which is difficult for large N, we can instead learn a Σ_K-equivariant model for $\mathcal{M}_{[K]}$, and "lift" it to achieve Σ_N-equivariance. We need to first define what is "good" binding, which intuitively means that we can project N objects to a subspace of K slots, while we are still able to identify them, i.e., how K-slot identities are permuted.

Definition 4.3 (Projection property)**.** Let h be an MDP homomorphism from the full MDP to the slot MDP, $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$. The homomorphism $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$ satisfies the projection property if, for any $\sigma \in \Sigma_N$, $s \in \mathcal{S}_{\mathbb{L}}$, $a \in \mathcal{A}_{\mathbb{L}}$, there exists $\overline{\sigma} \in \Sigma_K$ such that

$$\overline{\sigma}.\phi(s) = \phi(\sigma.s) \quad \text{and} \quad \overline{\sigma}.\alpha_s(a) = \alpha_{\sigma.s}(\sigma.a)\ . \tag{3}$$

In other words, the K present objects in $s \in \mathcal{S}_{\mathbb{L}}$ are *bound* to the K slots in $\phi(s) \in \mathcal{S}_{[K]}$ in a *specific* order. It implicitly assumes the binding of present objects in $\mathcal{M}_{\mathbb{L}}$ to slots in $\mathcal{M}_{[K]}$ is in some *temporally consistent* order: $\mathcal{S}_{i_k} \mapsto \mathcal{S}_k$.

By definition of MDP homomorphisms, given a model $\hat{T}_{\mathbb{L}}$ for $\mathcal{M}_{\mathbb{L}}$, h induces a model $\hat{T}_{[K]}$ for $\mathcal{M}_{[K]}$. If $\hat{T}_{\mathbb{L}}$ has equivariance error $\lambda_{\mathbb{L}}$, we can show that $\hat{T}_{[K]}$ will have the following equivariance error [2]:

Proposition 4.4. *Let $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$ be an MDP homomorphism satisfying the projection property. Suppose $\hat{T}_{\mathbb{L}}$ is a (learned) transition model of $\mathcal{M}_{\mathbb{L}}$ with equivariance error $\lambda_{\mathbb{L}}$. Then $\hat{T}_{[K]}$, the induced transition model of $\mathcal{M}_{[K]}$ under $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$, has sample equivariance error at $(\phi(s), \alpha_s(a), \phi(s')) \in \mathcal{S}_{[K]} \times \mathcal{A}_{[K]} \times \mathcal{S}_{[K]}$ and $\overline{\sigma} \in \Sigma_K$:*

$$\lambda^{\overline{\sigma}}_{[K]} \triangleq \Big[\Big|\hat{T}_{[K]}(\phi(s') \mid \phi(s), \alpha_s(a)) - \hat{T}_{[K]}(\overline{\sigma}.\phi(s') \mid \overline{\sigma}.\phi(s), \overline{\sigma}.\alpha_s(a))\Big|\Big] = C \cdot \lambda^{\sigma}_{\mathbb{L}}\ , \tag{4}$$

where $C = \binom{N}{K}$ is the number of K-slot scenes given an N-object library, $\phi : \mathcal{S}_{\mathbb{L}} \to \mathcal{S}_{[K]}$ and $\alpha_s : \mathcal{A}_{\mathbb{L}} \to \mathcal{A}_{[K]}$. The equivariance error is then $\lambda_{[K]} = \mathbb{E}_{s,a,s',\overline{\sigma}}[\lambda^{\overline{\sigma}}_{[K]}] = C \cdot \lambda_{\mathbb{L}}$.

The proof is provided in Appendix F. Therefore, if $\hat{T}_{\mathbb{L}}$ has perfect compositional generalization (equivariance error $\lambda_{\mathbb{L}} = 0$), and homomorphism h satisfies the projection property, then the induced model $\hat{T}_{[K]}$ is Σ_K-equivariant. Conversely, since the proposition holds with equality, if we have a Σ_K-equivariant model $\hat{T}_{[K]}$ in $\mathcal{M}_{[K]}$, and $\mathcal{M}_{[K]}$ is a homomorphic image of $\mathcal{M}_{\mathbb{L}}$, then we can lift the model to a Σ_N-equivariant model $\hat{T}_{\mathbb{L}}$ in $\mathcal{M}_{\mathbb{L}}$, which allows us to simplify Σ_N-equivariant models.

Corollary 4.5 (Lifted model is Σ_N-equivariant)**.** *If (1) $\hat{T}_{[K]}$ is Σ_K-equivariant, and (2) there exists a homomorphism $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$ satisfying the projection property, then $\hat{T}_{\mathbb{L}}$ is Σ_N-equivariant.*

4.4. Practically measuring compositional generalization

Even though we formally define the compositional generalization error as an expectation $\mathbb{E}_{s,a,s',\overline{\sigma}}[\lambda^{\overline{\sigma}}_{[K]}]$ over all

[2] However, from the computational perspective, the information of binding to latent space is unknown, so the numerical values are hard to compute, as detailed in Appendix C.

transition tuples and all permutations, directly computing it in *learned latent space* is not practical. The reason is that we only assume there exists correct object and action binding ϕ, α_s (by the projection property), while they are *learned* by models and *not* necessarily correct.

In the definition, several sources of error exist: (1) *prediction error* $(s, a) \to s'$, (2) *binding error* $(\phi(s), \alpha_s(a)) \to \phi(s')$, (3) *compositional error* $(\sigma.\phi(s), \sigma.\alpha_s(a)) \to \sigma.\phi(s')$. Alternatively, we need a practical metric to bypass (2) and measure (1)+(3). To this end, motivated by generating compositional natural language data (Keysers et al., 2020), we propose to train and test on disjoint set of scenes, and directly measure the prediction error on test scenes, which avoids explicitly "replacing" object in latent space $\sigma.\phi(s)$.

Specifically, we need to follow two rules analogously: (1) *Disjoint scene object set.* Training and test set of object scenes should be *disjoint*: $|\mathbb{O}| = K$, $\{\mathbb{O}_{\text{train}}\} \cap \{\mathbb{O}_{\text{test}}\} = \emptyset$. (2) *Similar object distribution:* $i_n \in \mathbb{L}$. Training and test datasets should have similar object frequency (uniform), to ensure the error is not biased by errors in object detection. The training set should also contain *all* objects in library $\bigcup \mathbb{O}_{\text{train}} = \mathbb{L}$. In the experiment section, we then use this strategy to collect data and measure prediction error on test set as a proxy of compositional generalization error.

5. Compositionally Generalizable WMs

In the previous section, we provided the mathematical foundations for achieving compositional generalization via Σ_N- and Σ_K-equivariant transition models in $\mathcal{M}_{\mathbb{L}}$ and $\mathcal{M}_{[K]}$ respectively. Now, we provide a practical approach, based on the construction in Section 4.3, for achieving soft Σ_N-equivariance in only Σ_K latent space. The overview of the model is provided in Figure 3.

We focus on *end-to-end* learning a world model in latent space. The key challenge is that the object binding is unknown and no canonical ordering can be determined purely from images, so the model needs to infer it from data. We assume the action space is factorized by objects (Guestrin et al., 2003; Kipf et al., 2019); further explained in Appendix B.2. While the factorization $\mathcal{A}_{\mathbb{L}} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ is fixed, the model must use interaction data (s, a, s') to *infer* which action controls which object.

Stage 1: Object Extraction. Object extraction, or object binding (Greff et al., 2020), learns object-structured representations, where each present object is represented by a latent vector $\mathbb{R}^D$, or *slot*. All slots form a latent state s_t, where no canonical order exists and the order differs in different time steps. This stage implicitly requires that *all* objects in $\mathbb{L}$ must have been observed during *training*; at generalization time we assume it can provide representations for any individual object. We use Slot Attention (Locatello et al., 2020) trained with reconstruction loss.

Stage 2: Action Binding. In learning the transition model $T(s, a) = s'$, we need to correctly align factorized actions a and object slots s, i.e., understand which action is controlling the object in each slot. This corresponds to the projection property required by the homomorphism $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$. We design an attention module, named **Action Attention**, that learns this purely from interaction (s, a, s'), as shown in Figure 3 (top right).

The attention matrix $M \in \mathbb{R}^{K \times N}$ is a (jointly learned) binding matrix from slots[3] s_t (at each time step) to some object identity, represented as an identity matrix $\text{Id}_{N \times N}$ in our case. This implements the *state-dependent* action transformation $\alpha_s(a)$, by *binding* K *actions* corresponding to the object slots: $\alpha_{s_t}(a_t) = M_t(s_t)a_t = \bar{a}_t$, and M_t is:

$$M_t(s_t) = \underset{K}{\text{softmax}}\left(\frac{1}{\sqrt{D}} k\left(\text{Id}_{N\times N}\right) \cdot q\left(s_t\right)^T\right), \quad (5)$$

where k, q are linear projections mapping to some dimension D, and a_t acts as the value v without further transformation. Intuitively, the object identity specifies a "name" to factorized actions and can be viewed as a preference for some canonical ordering of objects in the full MDP $\mathcal{M}_{\mathbb{L}}$.

Stage 3: Σ_K-equivariant Transition Modeling. Proposition 4.4 implies that we only need a Σ_K-equivariant transition function for modeling the slot MDP (instead of Σ_N for the full MDP), hence we use a message-passing GNN (Kipf et al., 2019). Crucially, we only require $K \ll N$ nodes, and since this leads to message-passing complexity of $O(K^2)$ instead of $O(N^2)$, we expect our model to be significantly more efficient. We also assume that interactions between any objects (slots) can be modeled by a one-edge network. This can be viewed as an OO-MDP (Diuk et al., 2008) with only one object class.

Training: Aligned Loss. The binding matrix $M_t \in \mathbb{R}^{K \times N}$ in Action Binding relates slots at time t to specific object identities. Thus, we use it to *align* slots s_t and s_{t+1} to the canonical full MDP space as $s_t^{\uparrow}$ and $s_{t+1}^{\uparrow}$ (Figure 3 (bottom right)) using the pseudoinverse: $s_t^{\uparrow} = M_t^{+}\bar{s}_t$, $s_{t+1}^{\uparrow} = M_{t+1}^{+}\bar{s}_{t+1}$, where $M_t^{+} = (M_t M_t^{\top})^{-1} M_t^{\top} \in \mathbb{R}^{N \times K}$, and $M_t^{+} T(\bar{s}_t, \bar{a}_t) \approx M_{t+1}^{+}\bar{s}_{t+1}$. We *jointly* train the model with the *aligned* loss between K slots, using the object-structured contrastive loss (Kipf et al., 2019); the positive part is:

$$\mathcal{L}^{+}(s_t^{\uparrow}, s_{t+1}^{\uparrow}) = \left\| \texttt{NG}(M_{t+1}^{+})\bar{s}_{t+1} - \texttt{NG}(M_t^{+})T(\bar{s}_t, \bar{a}_t) \right\|^2, \quad (6)$$

where $\texttt{NG}(\cdot)$ is *no-gradient* operation to prevent the shortcut to change the binding matrix M_t outside Action Attention. This forms a differentiable loss and allows us to train the

[3] In practice, we use $K + 1$ slots to also model the background, as in Slot Attention (Locatello et al., 2020).

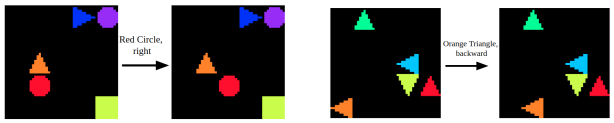

Figure 4: Example transitions of Shapes (left pair) and Rush Hour environment (right pair).

transition model in latent space, which we call **Homomorphic Object-oriented World Model** (HOWM).

Inference: Multi-step prediction in latent space. For multi-step prediction at inference for evaluation, illustrated in Figure 3, all slots need to be aligned with their corresponding actions with *learned* binding M_t, since their ordering is potentially different at all time steps. If aligning slots sequentially with learned M as $T(T(\bar{s}_t, \bar{a}_t), M_t M_{t+1}^{+} \bar{a}_{t+1}) \approx M_t M_{t+2}^{+} \bar{s}_{t+2}$ and compute difference, it would suffer from compounding error. Thus, we instead align *actions*: $T(s_t, M_{t+1}^{+} M_t a_t) = \hat{s}_{t+1}$, which gives preceding slots $\hat{s}_{t+1}$ (or telescoping for $t+k$) in the same order as s_t. More details in the Appendix D.

6. Experiments

We study how compositional generalization is achieved in practice, by using Object Library and generalization metrics. We provide additional results in Appendix G.

6.1. Experimental Setup

Environments. We designed two instances of the Object Library, ObjLib, environment. They are built upon the 2-D shape version of the Block Pushing environment (Kipf et al., 2019). (1) Basic **Shapes**. In the basic case, we equip the environment with a library of objects $\mathbb{L}$, where objects are different in shape, color, and size. The action space of each object is *identical*: `(north, south, east, west)`. (2) **Rush Hour**. To verify the importance of *action binding*, we set each object to have object-specific action space. This variant is motivated by the game *Rush Hour*, where each object is a car and has a (fixed) orientation (only using *triangles*). It can move relatively to its orientation: `(forward, backward, left, right)`. For example, if a car faces `east`, `right` will move `south`. For all variants, the action space is factorized by object and has *fixed* order across all scenes, as further discussed in Appendix B.2. More environmental setup is in Appendix H.4.

Methods. We study the compositional generalization (CG) performance of 6 approaches, under 3 categories:

1. Exact CG: Σ_N-equivariant methods with correct action binding should achieve perfect CG. One such method is $\mathbf{\Sigma_N}$**-CSWM**, the model from Kipf et al. (2019) with N object masks, one for each library object. Object masks and actions are bound by having the same ordering.
2. No guaranteed CG: To demonstrate the necessity of the three stages in Section 5, we consider methods that either do not have action binding or a Σ_N-equivariant transition model. In **C-WM(N)**, we break Σ_N-equivariance by replacing the N-slot GNN and shared encoder with a flat MLP. **MONet(N)+BM** builds on the model from Burgess et al. (2019), which only extracts objects into masks (slots); we use bipartite matching to align slots s_t and s_{t+1}, but *not* to actions, i.e., there is no action binding. We also include two variants of CSWM that uses only K slots (instead of N), but without an explicit binding mechanism: $\mathbf{\Sigma_K}$**-CSWM** assigns each of the K relevant action factors to slots (but the action may not actually control the object in the slot), whereas $\mathbf{\Sigma_K}$**-CSWM(CA)** makes all action factors available to all slots.
3. Soft CG: Our method, **HOWM**, achieves soft CG using a K-slot approach by relying on the learned action binding (see Section 5).

Additional details about the methods are in Appendix H.2.

Training and evaluation setup. We follow the setup in Kipf et al. (2019), using 1K episodes for training (consisting of 10 episodes of length 100, for 100 different scenes), and 10K episodes of length 10 for evaluation. Additionally, to evaluate compositional generalization, we ensure that (1) the combinations of objects in training dataset are different from those of evaluation dataset, and (2) the training data contains all N objects in the library. Additional training details for each method are in Appendix H.3.

Similar to Kipf et al. (2019), we measure the dynamics prediction error using two ranking metrics: Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR) (Kipf et al., 2019), averaged over 3 runs. For space, we only report MRR here; H@1 results are similar and can be found in Appendix G. We also report the difference between training and evaluation performance as an indication of the *generalization gap*; this is a proxy for equivariance error, which is difficult to compute in practice due to lack of ground-truth object binding information, as explained in Appendix C.

6.2. Results and analysis

We compare all methods on the Basic Shapes environment in terms of their generalization performance and scalability, shown in Table 1. With sufficient resources, Σ_N-CSWM should be the *upper bound* of performance, since it can achieve perfect CG, as verified by our results (near-perfect eval MRR, near-zero gap). Both Σ_K-CSWM and Σ_K-CSWM(CA) also achieve excellent training performance since they can memorize the correct action binding in each

Table 1: Results for all methods on the Shapes environment with $K = 5$ and $N = 5, 10, 20, 30$ (four numbers in each cell). "OM" stands for out of GPU memory, where we limit the usage to 10GB. We report for memory usage on $N = 20$.

CG Type	Env=Shapes	**Eval MRR** (%, 1-step)	**Eval MRR** (%, 5-step)	**Train MRR** (%, 5-step)	**Gap** (MRR %, 5-step)	Memory
(1. Exact CG)	Σ_N-CSWM	100. 100. 99.9 *OM*	99.9 99.9 99.9 *OM*	100. 100. 100. *OM*	0.0 0.0 0.1 *OM*	8.1GB
(2. No guaranteed CG)	Σ_K-CSWM	100. 80.4 70.8 74.1	99.9 43.7 32.2 29.4	100. 100. 100. 100.	0.1 55.3 67.8 70.6	1.5GB
	Σ_K-CSWM(CA)	95.0 72.0 71.4 80.9	85.0 26.8 22.7 24.3	96.3 96.5 96.1 98.0	11.3 69.7 73.4 73.7	1.6GB
	C-WM(N)	83.6 81.4 25.8 12.9	61.8 55.0 11.2 7.6	88.0 96.6 41.5 33.9	26.2 41.6 30.3 26.2	1.3GB
	MONet(N)+BM	12.6 73.9 35.9 *OM*	2.0 20.2 55.9 *OM*	7.0 64.5 84.8 *OM*	5.0 44.3 29.0 *OM*	9.3GB
(3. Soft CG)	**HOWM (ours)**	99.2 98.5 99.7 99.7	92.3 84.2 75.1 81.8	97.0 96.9 98.0 98.1	4.7 12.7 22.9 16.3	3.7GB

Table 2: Results on Rush Hour with relative action space, for $N = 5, 10, 20$ (three numbers in each cell).

Env=Rush Hour	**MRR** (1-step)	**MRR** (5-step)	**Gap** (MRR, 5-step)
Σ_N-CSWM	100. 100. 99.9	99.9 99.8 99.8	0.1 0.2 0.1
Σ_K-CSWM	100. 66.9 47.6	99.9 29.5 13.7	0.1 66.0 85.0
Σ_K-CSWM(CA)	99.2 55.4 80.2	94.9 15.5 15.8	4.2 84.5 84.2
C-WM(N)	55.5 67.8 80.6	23.7 24.5 45.3	48.8 70.7 54.1
HOWM (ours)	96.7 94.3 98.7	84.3 63.2 65.3	11.2 31.1 31.3

scene during training; however, when presented with new scenes in evaluation, the actions may not be bound correctly, leading to worse eval MRR and a large gap. C-WM(N) lacks Σ_N-equivariance and is significantly worse during both training and evaluation, even with more parameters. MONet(N)+BM performs even worse, once again indicating the importance of action binding. Interestingly, it seems to perform better with larger N, but also uses much more resources due to its Σ_N-equivariant model; like Σ_N-CSWM, it exceeds our memory budget for $N = 30$.

HOWM strikes a reasonable middle ground – the training performance (train MRR) is near-perfect, and generalization performance (eval MRR) is still quite high, though clearly not perfect. However, in contrast to all other methods except the Σ_N-CSWM upper bound, generalization is maintained as N increases, and the gap is significantly smaller. These results demonstrate that our proposed approach is able to achieve good generalization for intermediate $N \leq 20$ while consuming significantly fewer resources, and can scale to $N \geq 30$, unlike the exact Σ_N-CSWM method.

One limitation we observed is that for long-term prediction (≥ 5 steps), it is still quite challenging to achieve action binding and compositional generalization for a large library of objects. One reason is that the learned representation module (Slot Attention in our case (Locatello et al., 2020)) does not guarantee perfect object representations across time. This affects both action binding learning (if extracted slots are inaccurate, actions cannot be bound correctly) and long-term evaluation (if objects are missed at some step, predictions for all following steps are likely wrong).

In the Rush Hour environment, shown in Table 2, the action-binding problem is more challenging and requires more sophisticated modeling of object-dependent transition dynamics. We report the evaluation MRR and the generalization gap. As before, Σ_N-CSWM acts as an upper bound and performs near-perfectly with near-zero gap; however, as before, we expect that it will not scale beyond $N = 20$ with our resource budget. All other methods, including HOWM, have worse performance compared to Shapes. However, HOWM still generalizes significantly better to new scenes for $N = 10$ and $N = 20$, whereas performance of other methods declines more steeply.

6.3. Visualization

We visualize a model on a sampled transition (s_t, a_t, s_{t+1}) with $N = 10$ and $K = 5$ in Figure 5. Action attention matrices are computed using the states: $M_t(s_t), M_{t+1}(s_{t+1}) \in \mathbb{R}^{N\times(K+1)}$. The visualized M align with expectation: (1) K present objects are bound to K slots. (2) the absent $N - K$ objects are randomly assigned to slots (reasonable since not forced by loss), but they are consistent in t and $t + 1$ (as forced by loss to keep N-object $s_t^{\uparrow}, s_{t+1}^{\uparrow}$ embeddings close). Also, the lifted embeddings $M_t^+ \bar{s}_t, M_{t+1}^+ \bar{s}_{t+1}$ are well aligned and have little difference.

7. Related work

Object-oriented representations are crucial in artificial intelligence and robotics (Diuk et al., 2008; Wong, 2016); our framework is related to OO-MDPs (Diuk et al., 2008) consisting of a single class. Recently, a line of works studies learning to *discover* objects and their representations end-to-end. MONet (Burgess et al., 2019) applies sequential attention with VAEs to learn factorized representations. GSWM (Lin et al., 2020) builds generative models and learns end-to-end with variational inference. Slot Attention (Locatello et al., 2020) proposes attention at the pixel level, which can be viewed as soft clustering of pixels of objects. The order of object slots is decided by randomly initialized cluster centroids.

A object-oriented world model can be learned jointly or separately with object representation, using pixel reconstruction (COBRA (Watters et al., 2019), OAT (Creswell et al., 2021)), variational inference (STOVE (Kossen et al., 2019), OP3 (Veerapaneni et al., 2020)), or contrastive loss (C-SWM (Kipf et al., 2019), NPS (Didolkar et al., 2021)). Furthermore, in jointly learning object representations and world

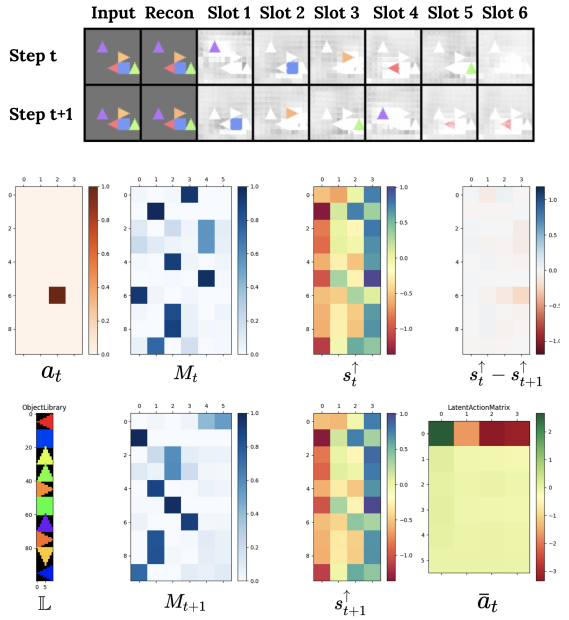

Figure 5: **(Top)** Visualization of learned slots; note that they are in random ordering. **(Bottom)** Components from a learned model for a transition, with $N = 10$ (corresponds to rows besides $\bar{a}_t$) and $K = 5$ (corresponding to 6 columns with an additional slot for background).

models, a key problem is *object binding* (Greff et al., 2020): binding object *slots* between time steps, which is typically solved by bipartite matching. In our setup, we emphasize the importance of correctly *binding actions* to objects. In video prediction such as (Kipf et al., 2021), action is not considered and thus has no action binding issue, which is our key focus and brings the major challenge. Biza et al. (2022) use location-based action space and learn attention to factorize monolithic actions. It keeps the setup in CSWM with $N = K$ and does not do compositional generalization in our $N > K$ formalism. We further discuss the choice of action space in Appendix B.2.

Compositional generalization has been widely studied in deep language models (Johnson et al., 2017). It also known as *systematic generalization* (Bahdanau et al., 2019), and *systematicity* (Lake & Baroni, 2018). Similar to us, Gordon et al. (2019) also use permutation equivariance to model compositional generalization in supervised learning on language data. Keysers et al. (2020) propose principles of measuring compositional generalization on sentences. A similar concept, *compositionality*, is usually referred to in disentangled representation learning. However, it typically focuses on disentanglement between individual attributes of distributed representations (Higgins et al., 2018; Caselles-Dupré et al., 2019), while we are more interested in factorization between (attributes of) objects. There is also work on measuring compositionality (Andreas, 2019; Chaabouni et al., 2020). Beyond language, Locatello et al. (2020); Veerapaneni et al. (2020) learn object-oriented representations, whose compositional generalization usually refers to factorization by objects. Furthermore, no prior work considers the difference between present objects K and its "library" N objects, i.e., they can only use Σ_N-equivariant model and thus struggle in scaling up.

Symmetries and MDP homomorphisms. Symmetry widely exists in various real-world data (Bronstein et al., 2021), and also in MDPs (Ravindran & Barto, 2004). A line of works in equivariant networks studies equivariance properties of existing network architectures, such as permutation equivariance in graph neural networks (GNNs) (Keriven & Peyré, 2019), while some other works study equivariant constraints to other symmetry groups (Bronstein et al., 2021). In RL, symmetries in MDPs can be modeled by MDP homomorphisms (Ravindran & Barto, 2004), which have nice properties such as *optimal value equivalence* and can be exploited in policy learning (van der Pol et al., 2020).

8. Conclusion

In conclusion, in this paper we introduced the Object Library family of object-oriented environments, as a first step toward studying compositional generalization in reinforcement learning. We defined compositional generalization in Object Library using the language of permutation equivariance, and showed a construction involving a K-slot MDP that was shown to achieve compositional generalization in Object Library. Based on this analysis, we introduced a three-stage pipeline for learning an object-oriented world model, and demonstrated in two instances of Object Library environments it could indeed generalize well to libraries with large N, while consuming fewer resources. Nevertheless, there remains a performance gap between our models and true compositional generalization that requires further study. Additionally, our analysis has been based on a fixed K, and assumes object persistence and a single object class, which clearly should be relaxed. Finally, while we have focused on learning transition models with compositional generalization, their application to planning and sequential decision-making should be explored.

Acknowledgments

This work was supported by NSF Grants #2107256 and #2134178. R. Walters is supported by The Roux Institute and the Harold Alfond Foundation. We also thank Ondrej Biza for helpful discussions and anonymous reviewers for useful feedback.

References

Andreas, J. Measuring compositionality in representation learning. In *International Conference on Learning Rep-*

resentations, 2019.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., and Courville, A. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2019.

Biza, O., van der Pol, E., and Kipf, T. The impact of negative sampling on contrastive structured world models. In *International Conference on Machine Learning Workshop in Self-Supervised Learning for Reasoning and Perception*, 2021.

Biza, O., Platt, R., van de Meent, J.-W., Wong, L. L., and Kipf, T. Binding actions to objects in world models. *arXiv preprint arXiv:2204.13022*, 2022.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. MONet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. Symmetry-based disentangled representation learning requires interaction with environments. In *Neural Information Processing Systems*, 2019.

Chaabouni, R., Kharitonov, E., Bouchacourt, D., Dupoux, E., and Baroni, M. Compositionality and generalization in emergent languages. In *Annual Meeting of the Association for Computational Linguistics*, 2020.

Creswell, A., Kabra, R., Burgess, C., and Shanahan, M. Unsupervised object-based transition models for 3d partially observable environments. *Advances in Neural Information Processing Systems*, 34:27344–27355, 2021.

Didolkar, A., Goyal, A., Ke, N. R., Blundell, C., Beaudoin, P., Heess, N., Mozer, M. C., and Bengio, Y. Neural production systems. *Advances in Neural Information Processing Systems*, 34:25673–25687, 2021.

Diuk, C., Cohen, A., and Littman, M. L. An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning*, 2008.

Gordon, J., Lopez-Paz, D., Baroni, M., and Bouchacourt, D. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*, 2019.

Greff, K., van Steenkiste, S., and Schmidhuber, J. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., and Lerchner, A. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

Johnson, J., Hariharan, B., Van Der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Keriven, N. and Peyré, G. Universal invariant and equivariant graph neural networks. In *Neural Information Processing Systems*, 2019.

Keysers, D., Schärli, N., Scales, N., Buisman, H., Furrer, D., Kashubin, S., Momchev, N., Sinopalnikov, D., Stafiniak, L., Tihon, T., et al. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2019.

Kipf, T., Elsayed, G. F., Mahendran, A., Stone, A., Sabour, S., Heigold, G., Jonschkowski, R., Dosovitskiy, A., and Greff, K. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

Kossen, J., Stelzner, K., Hussing, M., Voelcker, C., and Kersting, K. Structured object-aware physics prediction for video modeling and planning. In *International Conference on Learning Representations*, 2019.

Kroemer, O., Niekum, S., and Konidaris, G. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of Machine Learning Research*, 2021.

Lake, B. and Baroni, M. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, 2018.

Lin, Z., Wu, Y.-F., Peri, S., Fu, B., Jiang, J., and Ahn, S. Improving generative imagination in object-centric world models. In *International Conference on Machine Learning*, 2020.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. In *Neural Information Processing Systems*, 2020.

Ravindran, B. and Barto, A. G. *An algebraic approach to abstraction in reinforcement learning*. PhD thesis, University of Massachusetts at Amherst, 2004.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

van der Pol, E., Worrall, D., van Hoof, H., Oliehoek, F., and Welling, M. MDP homomorphic networks: Group symmetries in reinforcement learning. In *Neural Information Processing Systems*, 2020.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, 2020.

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P., and Lerchner, A. COBRA: Data-efficient model-based RL through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

Wong, L. L. *Object-based world modeling for mobile-manipulation robots*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2016.

Appendix

A. Outline

We discuss potential extensions and the choose of factorized action space in Section B. We provide some additional details of our framework for compositional generalization in Section E, followed by the proof of our main theoretical result, Proposition 4.4, in Section F. We further discuss the measure of compositional generalization in Section C and HOWM in D. Additional results, along with details about experiments and environments, are in Section G, Section H.4, and Section H, respectively.

B. Additional Discussion

B.1. Potential extensions

Our framework makes some assumptions in modeling the Object Library. For example, we assume object persistence, i.e., we have K specific objects in every scene. We can relieve this assumption, and thus needs to consider compositional generalization to different K's and objects changing in each scene. However, these assumptions are not fundamental limitations of the framework, and we provide some straightforward ideas to extend the framework.

Extension: reward and planning. Our framework can also handle reward. We do not include that, since we do not focus on using the learned model (in the slot MDP $\mathcal{M}_{[K]}$) for planning. Additionally, in other words, the model can also learn task-relevant features or objects that preserves the structure of reward and transition.

Extension: generalization to different K. Our framework can extend to compositional generalization to *different* K's. For a fixed K in the paper, there is a unique slot MDP $\mathcal{M}_{[K]}$ that is factored MDP and has permutation automorphism on it. However, for different K (usually training on smaller K's and generalizing to larger K's), there should be multiple slot MDPs for each K. Therefore, there would be an extra step to build a structured homomorphism between different slot MDPs first, $\mathcal{M}_{[K_i]}$ and $\mathcal{M}_{[K_j]}$. The model can then generalize to different K's by first generalizing to a slot MDP of the corresponding K_j (by recomposing structured homomorphism).

Extension: changing objects. The framework can also be extended to handle changing of objects during manipulation. We can split the episode to two episodes at the point of object changing. If the number of objects does not change, it is simply another scene. Otherwise, generalization to different values of K may need to be considered.

Extension: object class. In our framework, we implicitly assume that we just have one type of objects, or *class*, i.e. all interaction between objects can be modeled by one type of edge layer in graph neural networks. The concept of class is introduced in OO-MDPs (Diuk et al., 2008); our framework can be considered a single-class OO-MDP with multiple object instances of the same class. In contrast to OO-MDPs, our states have continuous attributes (instead of Boolean ones). Additionally, we perform representation learning from the image-based MDP $\mathcal{M}_{\text{pixel}}$, thus the learned object representation can be imperfect and more importantly, each individual slot can represent all objects and thus can represent different objects across time steps.

B.2. Factorized vs. location-based action space

In our work, we assume the action space is factorized by objects, similar to the state space. Empirically, this can be viewed as a high-level abstraction of the actions, such that we can independently control each object. However, there is another option: *location-based action space*, which chooses the object to control by location, like a touchscreen. We argue that it is inherently not a good choice for our setup. In the implementation, we require object-based world modeling to have (1) temporal consistency, and (2) compositional generalization. For location-based action space, these two requirements *cannot* be satisfied at the same time, even though location-based actions implicitly provide transition and thus have advantages.

Option (1) universal slots: temporal consistency issue. One option is that the extractor can bind any objects to any slots. However, this results in random slot ordering even between two adjacent time steps. Thus, the model needs to solve the correspondence between slots in two time steps. But because of location-based action space, there is no canonical ordering of either K present objects or K slots. In other words, the transition modeling has to rely on iterative approaches, such as bipartite matching, to optimize for a potentially correct order for every step, which is computationally expensive and does

not necessarily give correct answer, especially for long-term prediction.

Option (2) dedicated slots: compositional generalization issue. Another option is to fix the ordering in object extractor, so for every scene the extractor outputs object slots in *fixed* order. This is the case of our N-slot C-SWM variant. Unfortunately, this can only achieve perfect results in $N = K$ as the original paper does (Kipf et al., 2019), while it does not work for general case $N > K$, since there must be a slot that needs to bind to more than one object. However, we do *not* have a canonical ordering of all N objects in the library, so it is impossible to guarantee that for every seen or unseen scenes, it is able to provide dedicated and consistent slots across all time steps and all scenes.

If we do not consider computational efficiency, we can also feed actions of all objects to the transition model, i.e., no factorization of action. However, it becomes less meaningful for factorizing into N object slots and the model needs $O(N^2)$ complexity to model N object slots and N actions for every object slot.

Overall, factorized action space has several technical advantages and can be intuitively understood as a set of skills that independently control each object, which has been widely used in factored MDPs (Ravindran & Barto, 2004; Guestrin et al., 2003; Sutton & Barto, 2018) and robotics applications (Kroemer et al., 2021; Kipf et al., 2019; Veerapaneni et al., 2020; Biza et al., 2021).

C. Measuring Compositional Generalization in Object Library

By compositional generalization, we aim to generalize to novel object compositions with seen objects. This means that we train a model in slot MDP $\mathcal{M}_{[K]}$ using data from some scenes, and want to measure the performance on other scenes. In this process, one main error comes from the binding side $\mathcal{M}_{[K]} \mapsto \mathcal{M}_{\mathbb{O}}$: (1) from training scenes to the slot MDP, and (2) from the slot MDP to isomorphic novel scenes. Therefore, to estimate the performance, we need to have two set of scenes that are different compositionally, to fully measure the error.

To this end, we propose a strategy to generate training and evaluation data for dynamics prediction, which also appears similarly in generating compositional natural language data (Keysers et al., 2020). (1) *Disjoint scene object set:* $\{i_k : i_k \in [N], k \in [K]\} = \mathbb{O}_j \in \mathcal{P}_K(\mathbb{L})$. Scenes should sampled uniformly and different in training and evaluation set (not a trivial permutation). (2) *Similar object distribution:* $i_n \in \mathbb{L}$. Both datasets should have similar objects, to ensure the error is not biased by errors in object detection. In the experiment section, we use this strategy to collect data and measure the performance between $\hat{T}_{[K]}$ and $T_{[K]}$ directly in latent space.

Difficulty of numerically computing equivariance error. We use the above strategy to measure generalization performance in predicting action effects in novel scenes. Our theoretical framework technically provides another metric λ, Definition 4.2 in the main text, to measure compositional generalization. Although this metric is useful for proving our main theoretical result, it typically provides vacuous results for deterministic domains (e.g., the environments we use in our experiments), because the predicted transition function will always have an equivariance error of $\lambda = 1$, unless the transition function is exactly correct, only in which case $\lambda = 0$. Unfortunately, due to small prediction errors and numerical inaccuracies, empirically we always get $\lambda = 1$.

One alternative for deterministic domains is to instead compute the error in the (latent) *state* space (i.e., compare $\phi(s')$ vs. $\phi(\sigma.s')$), instead of in the transition probabilities. However, this raises another issue related to *binding*. In our slot-based framework, the ordering of slots is arbitrary, so the same dimensions of $\phi(s')$ and $\phi(\sigma.s')$ may not be comparable. Accurate comparison requires solving the binding problem, hence a metric computed in this fashion is approximate at best. Instead, we resort to simply measuring prediction performance using the strategy described above.

To the end, we chose to use *compositional generalization gap*, reported as a metric in the tables.

D. Additional Description of Homomorphic Object-oriented World Models

We extend the idea behind two proposed components: **Action Attention** and **Aligned Loss** to explain how the latent world model is able to train end-to-end, and also bring more details about **multi-step model prediction** in latent space which is not further elaborated in the main text.

The key challenge comes from the object-orientation representation side, where no canonical or unique order of objects can be defined in images. Thus, we have two potential paths to deal with this. One path is to use information from actions

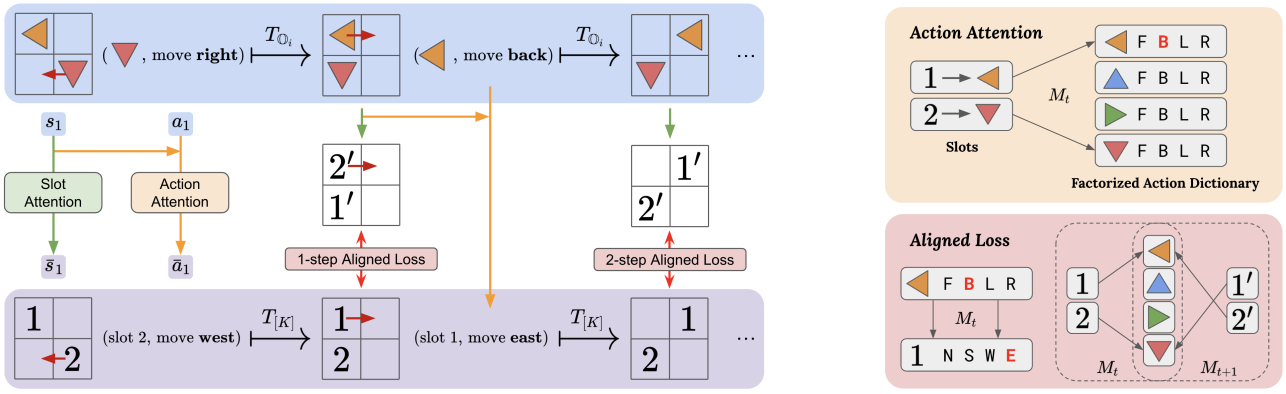

Figure 6: A copy of the figure in main text.

and the interaction with environment (s_t, a_t, s_{t+1}), since the action is usually independently controlling one or a few entities in the environment. Another path can be simply aligning the slots (in different order) across different time steps $(s_t, s_{t+1}, s_{t+2}, s_{t+3}, \ldots)$, which is the idea behind using bipartite matching or Hungarian algorithm and has been used in recent works for differentiable bipartite matching, for object detection and video understanding.

D.1. Contrastive Aligned Loss

We are based on the object-structured contrastive loss as described in (Kipf et al., 2019).

Since we have K slots and N possible objects in the library, a slot has to bind to more than one object. Thus, the order of the slots must be different for scenes with K objects from the library. Then, the problem is that, we cannot compute the loss directly between two slots in adjacent time step (in one action step away). To solve this without invoking bipartite matching between adjacent states (s_t, a_t, s_{t+1}), we need to make use of some canonical order. We assume there exists a fixed (across all scenes, guaranteed at data generation time in implementation) but unknown order of N objects in the library. Thus, there is an imaginary canonical full MDP, where the order of the factorized state and action space is fixed (but unknown and arbitrary).

The intuition to make use of this is straightforward: (1) in predicting the next state, we compute in the slot MDP $T(\bar{s}_t, \bar{a}_t)$ using Σ_K-equivariant GNN to lower the cost. (2) in computing the loss, we map the predicted latent states and target latent states (both represented in K slots but potentially in different order) back to the canonical full MDP, to ensure the temporally consistent order between adjacent steps (s_t, a_t, s_{t+1}). Intuitively, other unused $N - K$ "slots" (factorized state spaces) can be zeros.

In summary, this allows us to jointly learn (1) factorized (object-oriented) representations for N objects and (2) a latent Σ_K-equivariant transition model, by utilizing a factorized action space with fixed, arbitrary, unknown order.

Similar to (Locatello et al., 2020), the softmax is over K slots, where the intuition is that every object should select at most one slot. Thus, we also include an additional slot, for non-present objects and also for encoding background.

$$M_t(s_t) = \underset{K}{\text{softmax}}\left(\frac{1}{\sqrt{D}} k\left(\text{Id}_{N\times N}\right) \cdot q\left(s_t\right)^T\right) , \tag{7}$$

where k, q are linear projections mapping to some dimension D, and a_t acts as the value v without further transformation. Also, we only allow the gradient from Action Attention.

D.2. Multi-step Evaluation

We omit the details of multi-step evaluation in the main text, while this is the important component for how the model works for long-term prediction. The key idea behind multi-step evaluation is generic, where we learn a MDP homomorphism $h : \mathcal{M} \to \overline{\mathcal{M}}$, and unroll in the reduced MDP $\overline{\mathcal{M}}$ and somehow lift the prediction using learned model in $\overline{\mathcal{M}}$ back to the ground MDP $\mathcal{M}$. The main difference is that we only need the information of ordering, where the project property

guarantees that it is somehow preserved, thus the lifting is possible (only keeping the ordering correct, by aligning object slots to some canonical object order).

The key idea is to achieve prediction in latent space (slot MDP) $\bar{T}$, where for simplicity we use T below. We introduce how we achieve that. When we need to sequentially query the transition model to give us rollouts of the future in some latent slot space (i.e., the K-slot MDP), a problem comes: we map every image in a trajectory to a sequence of slots, while there is no mechanism enforcing temporal consistency, i.e., different slots in different time step can bind to different objects. So, we cannot use $T(...T(T(s_1, a_1), a_2), ...)$ to iteratively rollout the model directly. We propose a strategy that can rollout the model in latent space (the K-slot MDP) directly, without using bipartite matching or accumulated error of sequentially matching adjacent steps, under some assumptions: (1) we know that the factorized action space has a fixed but *unknown* order, and (2) the project property holds, i.e. the binding exists and object identities are preserved (we can identity how objects replacement corresponds to the permutation of slots).

The reason we can't align states directly is that, we need sequentially unrolling the transition model, which will result in using the binding matrix at each time step and easily have accumulated error (by multiple a sequence of learned approximate binding matrices).

Let's say $M_t M_{t+k}^+ = P_k \in \mathbb{R}^{K \times K}$, denoting the alignment to k steps ahead using learned binding M_t, which maps the slots to the canonical space (full MDP) and back to a specific K-slot ordering. If the project property satisfies, it should be full rank. Thus, for predicting next step, it is $T(\bar{s}_t, \bar{a}_t) \approx P_1 \bar{s}_{t+1}$, and similarly for $T(\bar{s}_{t+1}, \bar{a}_{t+1}) \approx P_{1\to 2} \bar{s}_{t+2}$.

Because of the Σ_K-equivariance property of the transition network, the prediction $M_t^+ T(\bar{s}_t, \bar{a}_t) \approx M_{t+1}^+ \bar{s}_{t+1}$ is equivalent to transforming actions: $M_{t+1} M_t^+ (T(\bar{s}_t, \bar{a}_t)) \approx \bar{s}_{t+1}$ and $T(\bar{s}_t, \bar{a}_t) \approx M_t M_{t+1}^+ \bar{s}_{t+1}$.

Similarly, for next step, we would need to align for the next step again:

$$T(M_t M_{t+1}^+ \bar{s}_{t+1}, M_t M_{t+1}^+ \bar{a}_{t+1}) = (M_t M_{t+1}^+)(M_{t+1} M_{t+2}^+) \bar{s}_{t+2} \tag{8}$$

$$\iff T(M_t M_{t+1}^+ \bar{s}_{t+1}, M_t M_{t+1}^+ \bar{a}_{t+1}) = (M_t M_{t+2}^+) \bar{s}_{t+2} \tag{9}$$

$$\iff T(T(\bar{s}_t, \bar{a}_t), M_t M_{t+1}^+ \bar{a}_{t+1}) = M_t M_{t+2}^+ \bar{s}_{t+2} \tag{10}$$

$$\iff T(T(\bar{s}_t, \bar{a}_t), P_1 \bar{a}_{t+1}) = P_2 \bar{s}_{t+2} \tag{11}$$

Telescoping to k steps ahead, we further get

$$T(\dots(T(\bar{s}_t, \bar{a}_t), (M_t M_{t+1}^+) \bar{a}_{t+1}) \dots, (M_t M_{t+k}^+) \bar{a}_{t+k}) = (M_t M_{t+k}^+) \bar{s}_{t+k} \tag{12}$$

$$\iff T(\dots(T(\bar{s}_t, \bar{a}_t), P_1 \bar{a}_{t+1}) \dots, P_k \bar{a}_{t+k}) = P_k \bar{s}_{t+k}. \tag{13}$$

Finally, we only need to map it to the N-object canonical full MDP:

$$M_t(T(\dots(T(\bar{s}_t, \bar{a}_t), (M_t^+ M_{t+1}) \bar{a}_{t+1}) \dots, (M_t^+ M_{t+k}) \bar{a}_{t+k})) = M_{t+k} \bar{s}_{t+k} \tag{14}$$

$$\iff M_t(T(\dots(T(\bar{s}_t, \bar{a}_t), P_1 \bar{a}_{t+1}) \dots, P_k \bar{a}_{t+k})) = M_{t+k} \bar{s}_{t+k}. \tag{15}$$

However, this requires us to align sequentially with $P_1, P_2, \dots, P_k$ for k-step prediction, which could not scale for large k. In actual implementation, we instead *align actions* to the order of s_t. Recall that the latent action is computed by $\bar{a}_t = M_t(s_t) a_t$, thus we have

$$(M_t M_{t+k}^+) \bar{a}_{t+k} = P_k \bar{a}_{t+k} = M_t a_{t+k}. \tag{16}$$

In computing the evaluation metric for multi-step prediction, such as k-step Hits and MRR, we could compute as follows:

$$M_t^+ \left(T(\bar{s}_t, [M_t a_t, M_t a_{t+1}, ..., M_t a_{t+k-1}])\right) \approx M_{t+k}^+ \bar{s}_{t+k}, \tag{17}$$

where $[M_t a_t, M_t a_{t+1}, ..., M_t a_{t+k-1}]$ means that we sequentially input $k-1$ actions to the world models and repressively predict next state using last state for k times. Intuitively, we do not need to know intermediate binding M_{t+1} through M_{t+k-1} since the actions are in a canonical order (defined by the N-object full MDP) and we align all intermediate slots to the slot order of first state s_t.

We use slot size of 16 and embedding size of 4 for all reported numbers. The negative states are sampled half from scenes with same objects $\mathbb{O}_i$, and half from different scenes, where we empirically find it works better than using purely random episodes, as further explained in Section G.

E. Details of the Compositional Generalization Framework

Symmetry widely exists in various real-world data (Bronstein et al., 2021), and also in MDPs (Ravindran & Barto, 2004). A line of works in equivariant networks studies equivariance properties of existing network architectures, such as permutation equivariance in graph neural networks (GNNs) (Keriven & Peyré, 2019), while some other works study equivariant constraints to other symmetry groups (Bronstein et al., 2021). The framework of *MDP homomorphism* is an algebraic approach studying *symmetries* and *abstraction* in reinforcement learning (Ravindran & Barto, 2004). MDP homomorphisms can be *induced* by symmetric stucture in MDPs (Ravindran & Barto, 2004; van der Pol et al., 2020), such as spatial transformations like reflections. We use the framework to study object replacement symmetry for formulating compositional generalization. Homomorphic MDPs have other nice properties to be further explored, such as *optimal value equivalence* (Ravindran & Barto, 2004). Also, the lifted policy learned in homomorphic MDPs has equivariance properties, which can be exploited by equivariant networks (van der Pol et al., 2020).

E.1. Definition with Reward

In the main paper, we focus on the equivariance of *transition* function. We provide the full definition of *error* also with *reward* function, another condition in MDP homomorphisms. The error of reward function only needs *invariance*, since it is a mapping with real-valued output: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Definition E.1 (Invariance error of $\hat{R}_{\mathbb{L}}$ on $\mathcal{M}_{\mathbb{L}}$)**.** Let $\hat{R}_{\mathbb{L}} : \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}} \to \mathbb{R}$ be a (learned) reward model of $\mathcal{M}_{\mathbb{L}}$. The invariance error λ_R of $\hat{R}_{\mathbb{L}}$ with respect to operation $p_{\mathbb{L}}$ is defined as:

$$\lambda_R \triangleq \mathop{\mathbb{E}}_{\sigma \in \Sigma_N, (s,a) \in \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}}} \left[\left| \hat{R}_{\mathbb{L}}(s,a) - \hat{R}_{\mathbb{L}}(\sigma.s, \sigma.a) \right| \right] , \tag{18}$$

where Σ_N is the permutation group of size N, and $\mathbb{L}$ is the object library of size $|\mathbb{L}| = N$.

The definition of the error on reward only needs invariance. The extended version of Proposition 4.4 including reward and its proof directly follow the one for transition function.

E.2. Definition of Induced Operations

We formally define the induced operation on scene and illustrate it with diagrams.

Definition E.2 (Object replacement operation $p_{\mathbb{L}}$)**.** Let Σ_N denote the permutations of N elements, which acts naturally on the object library set $\mathbb{L}$. The object replacement operation is the group operation $p_{\mathbb{L}} : \Sigma_N \times \mathbb{L} \to \mathbb{L}$ where Σ_N acting on $\mathbb{L}$ to replace the identity of objects.

For example, using disjoint cycle notation, an object replacement operation $\pi = (123)(45)$ acts on object indices in $\mathbb{L}$ by sending $\pi(3) = 1$ or $\pi(4) = 5$. Intuitively, this operation permutes the label or index of objects in $\mathbb{L}$, thus the scene has identical appearance but with object labels being replaced. This induces a group operation on scenes, which can be defined below.

Definition E.3 (Induced scene set operation p_K)**.** The permutation Σ_N on object library $\mathbb{L}$ *induces* an operation transforming scene object sets $\mathbb{O} \subset \mathbb{L}$. The induced group operation $p_K : \Sigma_N \times \mathcal{P}_K(\mathbb{L}) \to \mathcal{P}_K(\mathbb{L})$ acts on the set $\mathcal{P}_K(\mathbb{L})$ as such $p_K(\pi, \{i_1, \dots, i_K\}) = \{\pi(i_1), \dots, \pi(i_K)\}$, where $\pi \in \Sigma_N$, object indices $\{i_1, \dots, i_K\}, \{\pi(i_1), \dots, \pi(i_K)\} \in \mathcal{P}_K(\mathbb{L})$, each object index $i_k \in [N], K < N$.

Since i_k is the index of objects, p_K maps a set of objects to another set indexed by $\{\pi(i_1), \dots, \pi(i_K)\}$. If $\pi = (123)(45), K = 2$, then $p_K(\pi, \{3, 4\}) = \{1, 5\}$. The action of Σ_N on $\mathcal{M}_{\mathbb{L}}$ defined in Section 3.1 is compatible with the action of Σ_N on $\mathcal{P}_K(\mathbb{L})$ as $\sigma o(s) = o(\sigma s)$. Thus, Σ_N permutes the scene MDPs $\{\mathcal{M}_{\mathbb{O}}\}$. The induced operation on scenes implies some symmetry between sub-MDPs of scenes $\mathcal{M}_{\mathbb{O}}$ consisting of objects $\mathbb{O}$, which can be naturally studied by *MDP homomorphisms* between scene MDPs $\{\mathcal{M}_{\mathbb{O}} : \mathbb{O} \in \mathcal{P}_K(\mathbb{L})\}$.

E.3. Illustrative Examples

Graphically, the induced operation on scene can be seen as below.

Before introducing permutation groups, we first introduce how to use cycle notation to represent permutations. Assume our object library $\mathbb{L} = \{$▲, ▼, ◄, ►$\}$. The permutation in Figure 7 left is denoted by the cycle notation (▲◄)(▼►). The

Figure 7: Examples of permutations on a library $\mathbb{L} = \{\blacktriangle, \blacktriangledown, \blacktriangleleft, \blacktriangleright\}$.

permutation in Figure 7 right is denoted by the cycle notation $(\blacktriangle\blacktriangledown)(\blacktriangleleft\blacktriangleright)$. A permutation can be seen as an action that rearranges a set of elements. A permutation group of a set A is a set of permutations of A that forms a group under function composition.

We can use the symmetric group Σ_N to replace any object in the scene $\mathbb{O}_i$. For example, assume $N = 4$, $K = 2$ and scene $\mathbb{O}_i$ consists of objects $\{\blacktriangleleft, \blacktriangledown\}$. Then the group operation on $\mathbb{L}$ $((\blacktriangledown\blacktriangleright), \{\blacktriangle, \blacktriangledown, \blacktriangleleft, \blacktriangleright\}) \mapsto \{\blacktriangle, \blacktriangleright, \blacktriangleleft, \blacktriangledown\}$ is essentially doing the object replacement: σ ($\blacktriangledown$) = $\blacktriangleright$. Also, it can be viewed as scene replacement: R: $\{\blacktriangleleft, \blacktriangledown\} \mapsto \{\blacktriangleleft, \blacktriangleright\}$.

F. Proof of Proposition 4.1: Scaled Equivariance Error

We provide the proof of Proposition 4.4 in this subsection.

Although we have expectation in the definition, the relationship holds *point-wisely*, i.e., for every permutation $\sigma \in \Sigma_N$, the equivariance error is magnified by some constant $C = \binom{N}{K}$ in the slot MDP. We denote the error quantity of a permutation σ as $\lambda^{\overline{\sigma}}_{[K]}$ (for slot MDP) and $\lambda^{\sigma}_{\mathbb{L}}$ (for full MDP).

Note that the sample equivariance error is

$$\lambda^{\overline{\sigma}}_{[K]} =^{(1)} \left|\hat{T}_{[K]}(\phi(s') \mid \phi(s), \alpha_s(a)) - \hat{T}_{[K]}(\bar{\sigma}.\phi(s') \mid \bar{\sigma}.\phi(s), \bar{\sigma}.\alpha_s(a))\right|, \tag{19}$$

where $\bar{\sigma} \in \Sigma_K$ a permutation acting on $\mathcal{S}_{[K]}$, $\mathcal{A}_{[K]}$, and $\mathcal{S}_{[K]} \times \mathcal{A}_{[K]}$.

Since we assume the projection property holds: $\bar{\sigma}.\phi(s) = \phi(\sigma.s), \bar{\sigma}.\alpha_s(a) = \alpha_{\sigma.s}(\sigma.a)$, the equation holds:

$$\hat{T}_{[K]}(\bar{\sigma}.\phi(s') \mid \bar{\sigma}.\phi(s), \bar{\sigma}.\alpha_s(a)) = \hat{T}_{[K]}(\phi(\sigma.s') \mid \phi(s), \alpha_{\sigma.s}(\sigma.a)). \tag{20}$$

Thus the quantity can be transformed point-wisely as:

$$\lambda^{\sigma}_{[K]} =^{(2)} \left|\hat{T}_{[K]}(\phi(s') \mid \phi(s), \alpha_s(a)) - \hat{T}_{[K]}(\phi(\sigma.s') \mid \phi(s), \alpha_{\sigma.s}(\sigma.a)\right|. \tag{21}$$

By the definition of homomorphism ($\bar{T}$ is $\hat{T}_{[K]}$, and T is $\hat{T}_{\mathbb{L}}$) for transition function:

$$\bar{T}(\phi(s') \mid \phi(s), \alpha_s(a)) \triangleq \sum_{s'' \in \phi^{-1}(\phi(s'))} T(s'' \mid s, a), \forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}, \tag{22}$$

we substitute the equality

$$\lambda^{\sigma}_{[K]} =^{(3)} \left| \sum_{s'' \in \phi^{-1}(\phi(s'))} \hat{T}_{\mathbb{L}}(s'' \mid s, a) - \sum_{s'' \in \phi^{-1}(\phi(s'))} \hat{T}_{\mathbb{L}}(\sigma.s'' \mid \sigma.s, \sigma.a) \right|. \tag{23}$$

By further manipulating the quantity, we get

$$\lambda^{\sigma}_{[K]} =^{(4)} \sum_{s'' \in \phi^{-1}(\phi(s'))} \left|\hat{T}_{\mathbb{L}}(s'' \mid s, a) - \hat{T}_{\mathbb{L}}(\sigma.s'' \mid \sigma.s, \sigma.a)\right| =^{(5)} C \cdot \lambda^{\sigma}, \tag{24}$$

where the sample equivariance error in the full MDP $\lambda^{\sigma} \triangleq \left|\hat{T}_{\mathbb{L}}(s' \mid s, a) - \hat{T}_{\mathbb{L}}(\sigma.s' \mid \sigma.s, \sigma.a)\right|$ is defined in Definition 4.2, and $|\phi^{-1}(\phi(s'))| = \binom{N}{K}$ is equal for all s', since every combination can come from $\binom{N}{K}$ possible scenes, and we denote it as C. Finally, we apply expectation over all $\sigma \in \Sigma_N$ and arrive at the Proposition 4.4.

Table 3: Results for all methods on the Shapes environment with $K = 5$ and $N = 5, 10, 20, 30$ (four numbers in each cell). 'OM' stands for out of GPU memory, where we limit the usage to 10GB. We report for memory usage on $N = 20$.

Shapes	**MRR** (%, 1-step)	**H@1** (%, 5-step)	**MRR** (%, 5-step)	**Train** (MRR, 5-step)	**Gap** (MRR, 5-step)
Σ_N-CSWM	**100, 100, 99.9**, *OM*	**99,9, 99.8, 99.8**, *OM*	**99.9, 99.9, 99.9**, *OM*	**100, 100, 100**, OM	0.0, 0.0, 0.1, *OM*
Σ_K-CSWM	100., 80.4, 70.8, 74.1	99.8, 32.7, 22.6, 20.1	99.9, 43.7, 32.2, 29.4	100., 100., 100., 100.	0.1, 55.3, 67.8, 70.6
Σ_K-CSWM(CA)	95.0, 72.0, 71.4, 80.9	77.5, 18.7, 15.0, 15.4	85.0, 26.8, 22.7, 24.3	96.3, 96.5, 96.1, 98.0	11.3, 69.7, 73.4, 73.7
C-WM(N)	83.6, 81.4, 25.8, 12.9	49.6, 41.0, 4.4, 2.2	61.8, 55.0, 11.2, 7.6	88.0, 96.6, 41.5, 33.9	26.2, 41.6, 30.3, 26.2
MONet(N)+BM	12.6, 73.9, 35.9, *OM*	4.5, 11.5, 44.4, OM	2.0, 20.2, 55.9, *OM*	7.0, 64.5, 84.8, *OM*	5.0, 44.3, 29., *OM*
HOWM (ours)	96.7, 94.3, 98.7, 99.6	76.8, 51.7, 54.8, 35.7	84.3, 63.2, 65.3, 47.7	95.5, 94.3, 95.3, 94.7	11.2, 31.1, 31.3, 43.5

Table 4: Results for all methods on the Rush Hour environment with $K = 5$ and $N = 5, 10, 20$ (three numbers in each cell). 'OM' stands for out of memory for GPU training, where we limit the memory usage to 10GB.

Rush Hour	**MRR** (%, 1-step)	**H@1** (%, 5-step)	**MRR** (%, 5-step)	**Train** (MRR, 5-step)	**Gap** (MRR, 5-step)
Σ_N-CSWM	**100, 100, 99.9**	**100, 100, 99.9**	**99,9, 99.8, 99.8**	**100, 100, 99.9**	**0.01, 0.02, 0.01**
Σ_K-CSWM	100. , 66.9, 47.6	99.8, 18.9, 8.	99.9, 29.5, 13.7	100. , 95.1, 99.	0.01, 66.0, 85.0
Σ_K-CSWM(CA)	99.2, 55.4, 80.2	91.8, 8.7, 8.7	94.9, 15.5, 15.8	99.1, 100. , 100.	4.2, 84.5, 84.2
C-WM(N)	55.5, 67.8, 80.6	8.5, 13.4, 36.4	23.7, 24.5, 45.3	72.5, 95.2, 99.5	48.8, 70.7, 54.1
HOWM (ours)	99.2, 98.5, 99.7	88.9, 77.6, 66.3	92.3, 84.2, 75.1	97.0, 96.9, 98.0	4.7, 12.7, 22.9

G. Additional results

(HOWM) The results of HOWM show that our soft approach, with Action Attention module, is able to (1) end-to-end learn the approximately correct binding between K slots and N factorized actions and (2) correctly align adjacent time steps for computing contrastive loss, only through transition data (s, a, s') with the differentiable Aligned Loss. Furthermore, recall the multi-step evaluation, our approach unrolls the model in the latent K-slot MDP $\bar{s}_{t+1} = T(\bar{s}_t, \bar{a}_t)$ and aligns with the target states $s^{\uparrow}_{t+k} \approx s_{t+k}$, which only requires Σ_K-equivariance (thus $O(K^2)$ complexity). The results demonstrate that this latent approach is able to achieve reasonable results with Σ_N exact methods for intermediate $N \leq 20$ while consumes much less resources, and still has potential to scale to $N = 30$ and more. However, we found for long-term prediction (more than 5 steps), it is still quite challenging to achieve (1) action binding and (2) compositional generalization for large library perfectly, as $N = 30$ for 5-step shows. One potential reason is that, the learned representation module (Slot Attention in our case) does not guarantee perfect object representations for long steps. This seems to affect the learning of action binding (if slots are inaccurate, actions cannot be bound correctly) and the long-term evaluation (if some objects missed at some step, all following steps can be wrong).

(Representation bottleneck) In general, the learned object representation seems to be a main bottleneck of the decoupled training. For example, if Slot Attention (Locatello et al., 2020) cannot reliably separate and segment objects into slots, our downstream Action Attention module would struggle in binding actions with corresponding objects. Once a good representation module is learned, we found our approach has very small variance in transition learning. Also, the variance seems to highly depend on the number of library objects N. We choose the best representation checkpoints to train, but we found $N = 30, 50$ can succeed in most runs, while $N = 10, 20$ is likely to miss objects in $1/2$ or $1/3$ of runs.

(Training resource) We highlight the consumed resources in training Σ_N-equivariant models. As our framework suggests, these models should provide perfect compositional generalization, if it also has *action binding*. However, in practice, Σ_N-equivariant models require more training resources and cannot scale up. In Shapes, both MONet+BM and Σ_N-CSWM can only scale to around $N \approx 20$. For MONet+BM, we train representation with N object slots and Σ_N transition model separately, and each of them may take around 10GB for $N = 20$. For large N's, the training time of MONet+BM is approximately two days using one GPU. Σ_N-CSWM jointly learns N dedicated object masks, corresponding to the ordering of N factorized actions. We do note that, if there is enough resource available for N dedicated object masks, this approach provides the best available solution. It does not seem to be an universal approach because of independently training N object masks, where N is the number of **all** possible library objects that can possibly appear in an environment.

(Comparison between training with controlled and non-controlled negative sample) We change the negative sampling strategy. In particular, we select half of negative samples from the same object combinations and the other half from different object combinations. Figure 8 shows the impact of the controlled negative sampling strategy. With controlled negative sampling strategy, the performance of Σ_N-CSWM is improved by a large extent. We argue that the Σ_N-CSWM is inclined

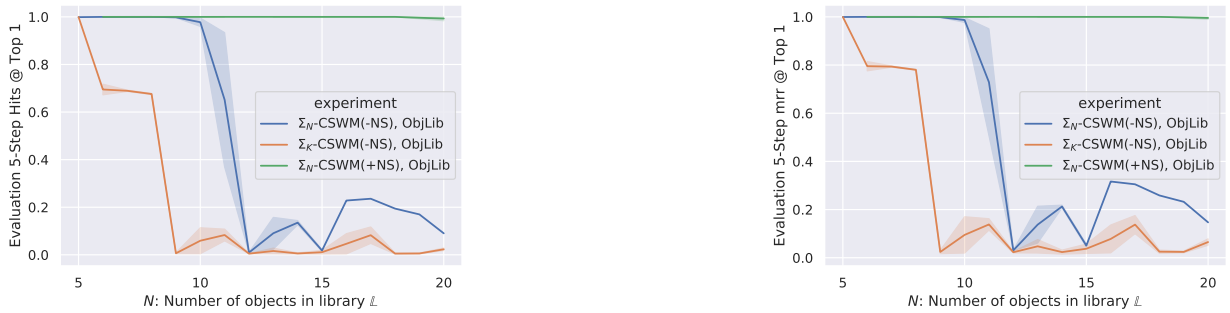

Figure 8: Comparison of different negative sampling strategy. Left figure shows the result of H@1 step 5, and right figure shows the result of MRR step 5.

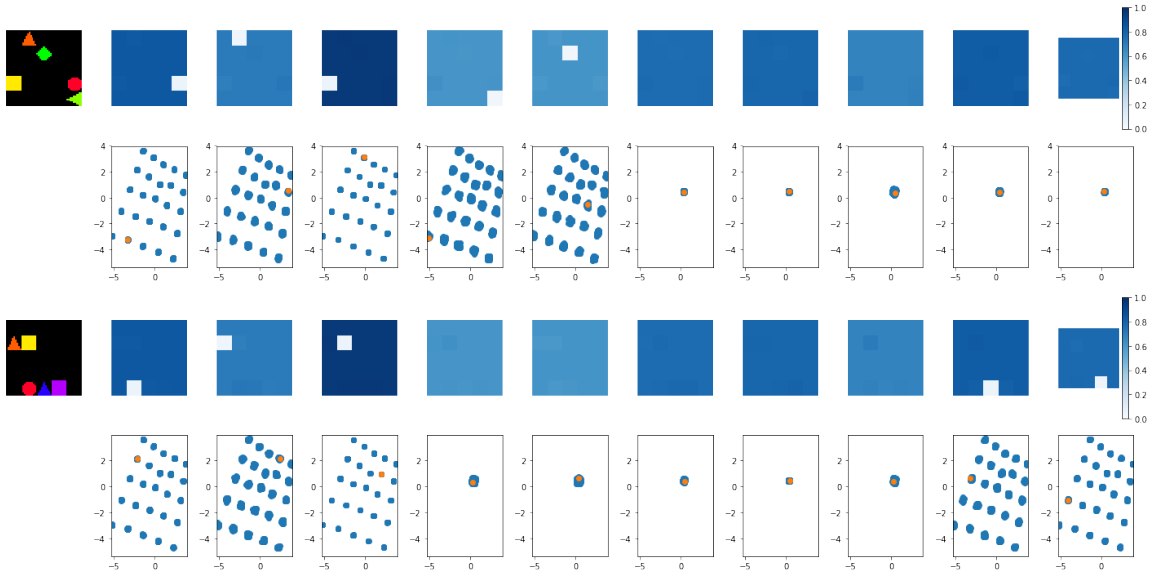

Figure 9: Learned object masks and embeddings in Shapes in two *unseen* evaluation scenes for Σ_N-CSWM (Exact CG). Present objects share similar latent structure and the model shows object replacement symmetry. Blue dots are from each individual object.

to go to local minimum without controlled negative sampling strategy.

(Negative sampling issue) We note for a sampling issue existing in two perspectives: (1) negative sampling in contrastive loss, (2) sampling reference states in computing Hits and MRR. The issue in negative sampling in contrastive loss used in (Kipf et al., 2019) is that, since the number of possible scenes $\binom{N}{K}$ increases greatly with N, if we train using randomly sampled negative states in contrastive loss (Kipf et al., 2019), we found the model will most likely sample negative states from other episodes, thus it may find a degenerate solution that just classifies scenes instead of objects in next states, as observed in (Biza et al., 2021). To fix the issue, in all training, we use 50% negative states from same object combinations and the other 50% from different object combinations. For the reference states in computing Hits and MRR, it needs to keep enough number of states from each scene to ensure each scene has enough cover, and also enough scenes to ensure diversity.

H. Additional Experiment Details

We experiment several approaches that can (2) learn object representations from images and (2) then object-structured world models. We further discuss their compositional generalization ability in the next section. **C-SWM** (Kipf et al., 2019) jointly learns object representations and GNN transition model with contrastive loss. We experiment two variants: (1) Σ_K-CSWM with K nodes in GNN (nubmer of objects on a scene), and (2) Σ_N-CSWM with N nodes (N = size of library). **MONet**

(Burgess et al., 2019) learns object representations using sequential attention and VAE to decompose scenes to multiple slots. We implement a N-slot version and applies bipartite matching for learning a GNN transition model on top of it, where we name it as MONet(N)+Σ_N+BM. **(Resource issue)** MONet(N)+Σ_N+BM and Σ_N-CSWM consumes CPU and GPU memory intensively, so we can only have small N. We also tried **STOVE** (Kossen et al., 2019), which uses RNN with depth of N and requires even more memory, thus cannot finish training for even $N = K = 5$.

H.1. Compositional generalization in the approaches

We experiment several approaches that can learn object representations from images and a latent object-structured world models. We classify the approaches based on how well they **should** achieve compositional generalization ("CG") and analyzing how they achieves CG.

(1, Exact CG) The first class should achieve exact CG under our framework, which is the Σ_N-equivariant methods with correct action binding. This only includes Σ_N-CSWM, which is a variant that outputs N object masks and has N nodes in its transition GNN. The way it achieves CG relies on learning N *dedicated* object masks, whose order is decided by N factorized actions. In other words, it *implicitly* bind objects and actions correctly, by fixing the order of object masks. This approach is **not** universal, since this requires N *dedicated* object masks (slots).

(2, Not guaranteed CG) If a method misses any of the necessary conditions (in three steps), it cannot achieve (perfect) CG. We leave (1) object extraction step untouched and focus on (2) action binding and (3) Σ_N-equivariant transition model. MONet(N)+BM uses bipartite matching in optimizing transition loss $\texttt{BM}(T(s,a),s')$. However, since it does *not* have any *action binding* mechanism, the transition network $T(\tilde{s},\tilde{a})$ has no clue to match object slots $\tilde{s}$ and factorized actions $\tilde{a}$. To break Σ_N-equivariance, we simply replace the N-slot GNN and shared encoder with a flat MLP, named C-WM(N). We also use two variants of CSWM with K slots, where Σ_K-CSWM receives *factorized* actions only from K scenes, and Σ_K-CSWM(CA) means we copy actions from N objects to all K slots, since no action binding is deployed.

(3, Approx. CG) As comparison, HOWM achieves soft compositional generalization by learning binding objects and actions with *action attention*, which is easier to scale up to large N.

H.2. Implementation and training setup

(Data) In generating training and evaluation dataset, we guarantee that (1) the combinations of objects in training dataset are different from those of evaluation dataset, and (2) training data contains all N objects in the library. In practice, we found it is critical to keep all individual objects seen in training, or at evaluation the model would very likely mess up objects' colors or shapes, as further detailed in Appendix 4. We use the same object library for all experiments for lower randomness. We use 1k episodes in training and 10 episodes of length 100 for 100 scenes, and 10k episodes of length 10 for evaluation. Additionally, we make sure that 90% of the transitions have objects moving (filtering collisions), so there is denser signals for relating actions and moved objects.

(Training) For Σ_K-CSWM and Σ_N-CSWM, we jointly train the entire model (Kipf et al., 2019). We train MONet(N)+BM by first learning the object-oriented representation using pixel reconstruction loss, then freeze the representation module and train the GNN transition model with bipartite matching in the learned latent space. See more details in Appendix H.

(**Metrics**) We measure the (multi-step) dynamics prediction error using two ranking metrics: Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR) (Kipf et al., 2019), averaged over 3 runs. For our approach, we need to align them in the full MDP $\mathcal{M}_{\mathbb{L}}$. We also report these metrics in training scenes and compute the *generalization gap*, as a proxy for compositional generalization error, because of the lack of object binding information in latent space.

- Σ_K-CSWM and Σ_K-CSWM-CA. C-SWMs (Kipf et al., 2019) are Contrastively-trained Structured World Models that apply a contrastive approach to learn the representation and the transition model of environments with compositional structure. Specifically, the models learn a set of factorized state variables, which encodes information about each object in the scene. These state variables are then fed into to a graph neural network to model the transitions of the environment. We call this approach $\texttt{C-SWM}(K)$ because in the original paper, they do not consider scenes with different combinations of object. In other words, they only consider settings with $N = K = 5$, where the combination across different episodes remains the same. We first use the model in the original paper as a baseline, which we term Σ_K-CSWM. Then, we further adapt the model by concatenating the whole action vector to each object representation and then feeding them into the graph neural network, which we term Σ_K-CSWM-CA.

- Σ_N-CSWM. To further study the capability of C-SWMs, we increase the number of object slots to N, i.e., we use N slots to learn the full transition model of the object library. In the original C-SWM model, the number of object slot is equal to the number of objects in the scene, which is 5 in 2D shapes environment. We also increase the number the action to $4 \times N$, where each 4 actions control one specific object.

- C-WM(N). We use non-factorized encoder MLP and transition MLP as a baseline. The purpose of this baseline is to see how the most naive model will perform in terms of compositional generalization.

- MONet(N)+Σ_N+BM. MONet (Burgess et al., 2019) decomposes images into different slot variables, each of which represents information about objects or background. It applies a recurrent attention network to produce attention masks for objects and background, and then feed each mask together with the image to a component VAE to learn object-oriented representations. We divide the experiment into two phrases. First, we train the MONet model using the image input to learn the object-oriented representation. The hyperparameters of the original MONet do not work well on our dataset because models like MONet rely on inductive biases, and often need to be adjusted to new datasets. In particular, the component VAE is too flexible for the dataset so we have to reduce the dimension of the latent variables to 4. We further change the structure of the attention U-net and the encoder and decoder of the component VAE.

  Second, we freeze the model parameters of MONet and use the learned object-oriented representation as input of the GNN to learn the transition model of the environment. For MONet(N)+Σ_N+BM, we implement a bipartite matching function between the object representation model and the transition model.

- *Other related baselines*. Slot attention module proposed in (Locatello et al., 2020) is a component that connects the perceptual representations such as feature maps extracted by CNN and the object-based slot representations. It uses iterative attention to enable each slot to compete for explaining part of the perceptual input. Based on the original `C-SWM` model, we place the slot attention module between the feature maps extracted by CNN and the object-based slot representations, and the other parts of the model remain the same. STOVE (Kossen et al., 2019) uses RNN with depth of N, that require much memory, and cannot finish training for $N = 5$.

H.3. Training and Evaluation Setup

We match the training procedures with C-SWM (Kipf et al., 2019) where possible. For `C-SWM` baselines and our own model, we joint train the representation module and transition module, similar to C-SWM (Kipf et al., 2019). We train the MONet model using the image input to learn the object-oriented representation. We then freeze the model parameters of MONet and use the learned object-oriented representation as input of the GNN to learn the transition model of the environment. All models are trained on Nvidia GeForce RTX 2080 Ti GPU with 11GB memory.

- Σ_K-CSWM, Σ_N-CSWM, and C-WM(N). We follow the same training and evaluation settings as in (Kipf et al., 2019). In particular, for training dataset, we generate 1000 episodes, with each 100 time steps; for evaluation dataset, we generate 10000 episodes, with each 10 time steps. Models are trained for 100 epochs. We use Adam optimizer with a learning rate of 5×10^4 and batch size of 1024, margin of hinge loss $\gamma = 1.0$, same as the original paper.

- MONet(N)+Σ_N+BM. We divide the experiment into two stages: object-representation learning and transition model learning. The way we generate the training and evaluation dataset is the same as Σ_K-CSWM. For the first stage, we only use image observations for training. For MONet (Burgess et al., 2019) implementation, we adapt the code from a third-party implementation[4]. We follow the training hyperparameters suggested in the original MONet paper. We use the following settings: flags: `-geco False -pixel_std1 0.09 -pixel_std2 0.11 -train_iter 1000000 -batch_size 64 -optimiser rmsprop`. For the second stage, we use the same settings as Σ_K-CSWM.

H.4. Additional Environment Details

Our environment is built upon the 2D shapes environment in (Kipf et al., 2019). The original environment consists of 5 objects and the actions are discrete. Actions are moving individual object into four cardinal directions, and thus $|\mathcal{A}| = 5 \times 4$ discrete actions in total. At each time step, the agent can only take one action, i.e., moving one object to one specific direction. After the agent took an action, the corresponding object would move into the direction by one step unless the

[4] https://github.com/applied-ai-lab/genesis

Figure 10: Example observations of OBJLIB. (**left**) $K = 5$. (**right**) $K = 3$.

target position is occupied by other objects or out of boundary. The agent uses a random policy to collect the experience buffer containing a sequence of tuple $\mathbb{B} = \{o_t, a_t, o_{t+1}\}$, where o_t is the observation at time step t, a_t is the action taken, and o_{t+1} is the next observation. Observations are $3 \times 50 \times 50$ RGB images containing K different objects. Each object occupies 10×10 pixels and each location of the overall 50×50 pixels can only be occupied by at most one object.

Based on this environment, we create an object library consisting of a certain number of objects, each of which has a unique shape and color combination. At each episode, we choose K objects from the library and place them to our environment. For training dataset, we randomly sample K objects from the object library except the consecutive sequences. For evaluation dataset, we sample one objects randomly, choose the subsequent consecutive K objects. In this way, we can guarantee that the combinations of objects of training dataset are totally different from those of evaluation dataset. We can also generate dataset with different Ks, as shown in Figure 10. We implement two versions of OBJLIB, named **Shapes** and **Rush Hour**.

For **Rush Hour**, the shapes of objects are chosen from four triangles heading to four different directions, while the colors are all different. The actions are not move up, down, left and right anymore. Rather, it depends on the heading of the triangle. We define: the heading of the triangle as move forward and the opposite direction as move backward; left of the heading as move left and right of the heading as move right. In other words, there are four different action spaces, each of which corresponds to the triangle heading to one specific direction. For example, if a triangle (car) faces east, then taking action `forward` results in moving east, `backward` west, `right` south, and `left` north; if a triangle (car) faces south, then taking action `forward` results in moving south, etc.

To evaluate compositional generalization of the model, i.e., the capability to understand novel combinations of known objects, we want the combination of objects of the train dataset and evaluation dataset to be as different as possible, while every individual object will appear in both dataset. One analogy of this is to think of it as a chemical experiment (Keysers et al., 2020). Each example (compound) is generated by combining primitive elements (atoms). We can view chemical experiments as rearranging atoms to generate new compounds. The atoms are contained in both train and evaluation dataset, while there are unseen compounds in evaluation dataset.

Generating data with controllable object combinations for negative samples. We generate data with controllable object combinations for negative samples. In particular, for training dataset, we select negative samples from the same object combination with probability ϵ and from different object combinations with probability $1 - \epsilon$. For training dataset, we only sample some combination of K objects; for evaluation dataset, we sample combinations that are different from each combination in training dataset. For the training dataset, we limit the number of different combinations to N, and generate $\#$of episodes$/N$ episodes for each combination. During data generation, we save the type of objects (color and shape), which are then used for all experiments.

I. Model Architectures

We provide the details of model architecture of the methods we use in experiment in several tables.

Σ_N-CSWM model architecture is described in below tables. Table 1 describes the object extractor and Table 2 describes the object Encoder. The transition model is GNN-based, where both the node and edge model are the same architecture as object Encoder.

Table 5: Object Extractor for Σ_N-CSWM baseline

type	size/channel	Activation	Comment
Conv 10 x 10	32	ReLU	Stride: 10
BatchNorm2d	-	-	-
Conv 1 x 1	N	Sigmoid	Stride: 1

Table 6: Object Encoder for Σ_N-CSWM baseline

type	size/channel	Activation	Comment
Linear	25 x 512	ReLU	-
LayerNorm	-	-	-
Linear	512 x 512	ReLU	-
LayerNorm	-	-	-
Linear	512 x 2	ReLU	-

Table 7: Monet attention downsampling

type	size/channel	Activation	Comment
Conv 3 x 3	32	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	32	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	64	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	64	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	64	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-

Table 8: Monet attention upsampling

type	size/channel	Activation	Comment
Conv 3 x 3	64	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	64	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	32	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	32	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-
Conv 3 x 3	32	ReLU	Stride: 1, Padding: 1
BatchNorm2d	-	-	-

Table 9: Monet attention baseline

type	size/channel	Activation	Comment
Linear 1600 x 128		ReLU	-
Linear 128 x 128		ReLU	-
Linear 128 x 1600		ReLU	-

Table 10: Monet baseline encoder

type	size/channel	Activation	Comment
Conv 10 x 10	32	ReLU	Stride: 10
BatchNorm2d	-	-	-
Conv 1 x 1	N	ReLU	Stride: 1
Flatten	-	-	-
Linear	N x 25 x 512	ReLU	-
Linear	512 x 512	ReLU	-
Linear	512 x 8	-	-

Table 11: Monet baseline decoder

type	size/channel	Activation	Comment
BroadcastLayer	-	-	-
Conv 3 x 3	16	ReLU	Stride: 1
Conv 3 x 3	16	ReLU	Stride: 1
Conv 1 x 1	4	-	Stride: 1