LEARNING LIFTED WORLD MODELS WITH TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work has studied whether transformers trained on sequential data learn reusable world models. In this paper, we address this question for *lifted models*, i.e., abstract world models that generalize across different numbers of objects. To this end, we leverage STRIPS, a logic-based formalism for specifying lifted, interpretable world models. We propose two transformer architectures for learning STRIPS models: the Lifted STRIPS (LS) Transformer, with built-in symbolic structure and parameters that map directly to STRIPS; and the Lifted Stick-Breaking (LB) Transformer, an architecture closer to a standard Transformer that replaces softmax with stick-breaking attention and uses randomized object embeddings. The transformers are trained in a supervised manner on action traces under two supervision settings. Under applicability supervision, each action in a sequence is labeled as applicable or inapplicable given the initial state and preceding actions, and no subsequent states are observed. Under final-state supervision, each trace contains an initial state, an applicable action sequence, and the resulting final state, while intermediate states remain unobserved. Experiments on five domains show that both transformers achieve high accuracy and generalize to longer traces and larger numbers of objects. Additionally, the extracted STRIPS models are fully interpretable and can be used for planning with off-the-shelf symbolic planners.

1 INTRODUCTION

Recent work has studied whether transformers trained for next-token prediction acquire world models (Toshniwal et al., 2022; Li et al., 2023; Nanda et al., 2023; Vafa et al., 2024; 2025) and generalize to longer sequences (Kazemnejad et al., 2023; Zhou et al., 2024; Huang et al., 2025a). Sarrof et al. (2026) study the related task of plan verification in the lifted STRIPS setting, where the domain is given and transformers must generalize both to longer plans and to larger numbers of objects. Finally, Núñez-Molina et al. (2026) use transformers to learn propositional STRIPS models from action sequences for use with off-the-shelf classical planners. The input traces contain ground actions over concrete objects, whereas the target model consists of object-independent action schemas.

In this work, we bring these different threads together by addressing the problem of *learning lifted* STRIPS *models* from suitable traces using a transformer architecture. Lifted STRIPS is a logic-based formalism for representing deterministic world models in terms of predicates and action schemas defined over variables, so that the same model can be instantiated over different sets of objects (Ghallab et al., 2004; Geffner & Bonet, 2013). A learned lifted model should therefore generalize not only to traces of different lengths, but also to problem instances with different numbers of objects.

We consider two supervision settings. Under *applicability supervision*, each trace contains an initial state and an action sequence in which every action is labeled as applicable or inapplicable given the initial state and the preceding actions, and no subsequent states are observed. Under *final-state supervision*, each trace contains an initial state, an applicable action sequence, and the resulting final state, while intermediate states remain unobserved. The first setting is related to the traces used by Núñez-Molina et al. (2026) to learn propositional STRIPS models, whereas the second is related to those used by Sarrof et al. (2026) for plan verification in lifted STRIPS.

Our main contribution is a pair of transformer architectures for learning lifted STRIPS models, extending Núñez-Molina et al. (2026) from propositional to lifted models. The *Lifted* STRIPS *(LS) Transformer* incorporates explicit symbolic structure, with parameters mapping directly to STRIPS

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

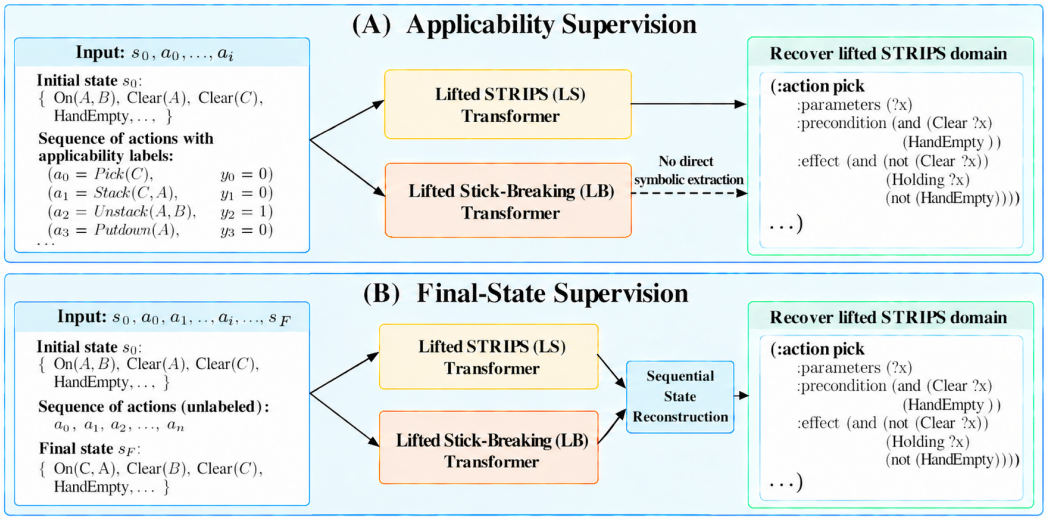


Figure 1: **Overview of the two architectures and supervision settings. (A) Applicability supervision:** each trace contains an initial state and an action sequence of applicable and inapplicable actions, without any subsequent states. In this setting, extracting a STRIPS model is possible only for the LS Transformer. **(B) Final-state supervision:** each trace contains an initial state, a sequence of applicable actions, and a final state, while intermediate states remain unobserved. A lifted STRIPS model can be extracted from either architecture through sequential state reconstruction.

preconditions and effects, and is based on a B-RASP formulation of lifted STRIPS trace recognition. The *Lifted Stick-Breaking (LB) Transformer* is closer to a standard Transformer, replacing softmax with stick-breaking attention (Tan et al., 2025), without positional encodings, and using randomized object embeddings. Under applicability supervision, we train the LS Transformer and extract the STRIPS model directly from its parameters. The LB Transformer could also be trained in this setting, but its parameters do not map directly to STRIPS, so we do not extract a symbolic model. Under final-state supervision, we train both architectures and recover the model through state reconstruction: the transformer predicts the unobserved intermediate states, and an algorithm extracts the lifted STRIPS model from the reconstructed state-action traces. Figure 1 illustrates both settings.

We evaluate the LS and LB Transformers on five benchmark planning domains. Under final-state supervision, both architectures achieve high accuracy, substantially outperforming standard Transformer baselines with softmax attention and positional encodings. Additionally, both architectures show strong generalization to traces up to ten times longer and, separately, to problem instances with up to ten times more objects than those seen during training. Finally, we extract lifted STRIPS models from both transformers and use them for planning with off-the-shelf symbolic planners. The extracted models are fully interpretable, allowing us to inspect the world dynamics learned by the transformers and identify precise errors when they occur.

2 Related Work

Transformers and World Models. Work on chess and Othello probes whether transformers trained for next-token prediction internally represent board states (Toshniwal et al., 2022; Li et al., 2023; Nanda et al., 2023). In maze solving, Spies et al. (2025) study representations of an input connectivity graph, while Rohekar et al. (2025) investigate causal structure in attention. Vafa et al. (2024) ask whether generative models recover the underlying transition structure, while Vafa et al. (2025) study whether predictive models acquire inductive biases that transfer to new tasks. Rather than only probing the implicit representations from the trained transformer, we recover explicit lifted STRIPS models. Another line learns symbolic world models for robot planning from perceptual observations, using neuro-symbolic abstractions and pretrained vision-language models (Liang et al., 2025; 2026; Athalye et al., 2026). In contrast, we learn without intermediate-state observations, and under applicability supervision no successor states are observed at all.

Formal Languages and Transformer Generalization. RASP and B-RASP are abstract languages for reasoning about soft- and masked hard-attention Transformers, respectively (Weiss et al., 2021; Yang et al., 2024). Masked hard-attention Transformers recognize exactly the star-free languages (Yang et al., 2024), a class that includes valid action sequences in a STRIPS domain (Lin & Bercher, 2022). More recently, C-RASP extended RASP with counting operations to study length generalization in soft-attention transformers (Yang & Chiang, 2024; Huang et al., 2025b). Sarrof et al. (2026) later extended C-RASP to C*-RASP to analyze lifted STRIPS plan verification with increasing plan lengths and numbers of objects. In contrast, we use B-RASP and learn lifted STRIPS models, with plan verification only as an intermediate step.

Learning Action Models for Planning. Action model learning has a long history in AI planning, from methods using fully or partially observed state-action sequences (Zhuo & Kambhampati, 2013; Aineto et al., 2019; Lamanna et al., 2025) to approaches recovering lifted rules from action traces alone (Cresswell et al., 2013; Gösgens et al., 2025). Recent work studies how little state and action information is needed to identify the underlying dynamics (Jansen et al., 2025; Gösgens et al., 2026), while neural methods have also been used to learn propositional and lifted action models (Asai et al., 2022; Xi et al., 2024; 2026; Ajith & Chamzas, 2026). Our work extends Núñez-Molina et al. (2026) from propositional to lifted STRIPS models and to learning from only applicable actions when final states are observed.

Pre-trained LLMs and Symbolic Planning Models. Recent work uses pre-trained LLMs to infer symbolic planning models from natural-language descriptions, demonstrations and simulations (Guan et al., 2023; Oswald et al., 2024; Yu et al., 2025; Huang et al., 2026; Hao et al., 2026; Xie et al., 2025). By contrast, our models are trained from scratch on symbolic traces and do not rely on pre-trained semantic knowledge.

3 BACKGROUND

3.1 STRIPS MODELS

A lifted STRIPS model or domain $M = \langle \mathcal{P}, A \rangle$ consists of a finite set of predicate symbols $\mathcal{P}$ and action schemas A (Russell & Norvig, 2010; Ghallab et al., 2004; Geffner & Bonet, 2013). Each predicate $p \in \mathcal{P}$ denotes a relation over objects with fixed arity $r = \mathrm{ar}(p)$. Each action schema $\alpha(x_1, \ldots, x_k) \in A$ specifies sets of preconditions $\mathrm{pre}(\alpha)$, add effects $\mathrm{add}(\alpha)$, and delete effects $\mathrm{del}(\alpha)$ whose elements are lifted atoms of the form $p(x_{\rho_1}, \ldots, x_{\rho_r})$. A valid pattern for α is a tuple $\rho = (\rho_1, \ldots, \rho_r) \in \{1, \ldots, k\}^r$ with pairwise distinct entries, specifying which action variable fills each argument of p. For $r = 0$, the unique pattern is $\rho = ()$. We restrict attention to action schemas whose lifted atoms do not repeat action variables. We use *lifted* to distinguish this setting from *propositional* STRIPS, whose atoms and actions are ground over a fixed object set. Throughout, STRIPS refers to the lifted setting unless stated otherwise. We use untyped STRIPS, encoding types as unary predicates.

A STRIPS problem or instance $P = \langle M, O, I, G \rangle$ consists of a STRIPS model M, a finite set $O = \{o_1, \ldots, o_N\}$ of N objects, an initial state I and a goal G. Instantiating predicate and action variables with object tuples $\mathbf{o} \in O^r$ and $\mathbf{u} \in O^k$ yields ground atoms $p(\mathbf{o})$ and ground actions $a = \alpha(\mathbf{u})$, with the corresponding grounded preconditions and effects. The initial state I and any state s are sets of true ground atoms, while G is a set of goal atoms. A ground action a is applicable in s when $\mathrm{pre}(a) \subseteq s$. When applicable, it yields $s' = (s \cup \mathrm{add}(a)) \setminus \mathrm{del}(a)$. A trace $\tau = (a_0, \ldots, a_{n-1})$ is applicable from $s_0 = s$ if each a_i is applicable in s_i and produces s_{i+1}. It is a plan for P if $s_0 = I$ and $G \subseteq s_n$. Finally, since M is lifted, the same model can define problems over different finite sets of objects, of arbitrary size.

3.2 TRANSFORMERS AND ATTENTION

A Transformer (Vaswani et al., 2017) maps token representations $(h_0, \ldots, h_{n-1})$ to contextual representations using self-attention and position-wise feed-forward layers. In an attention head, each representation is projected into a query $Q(i)$, key $K(i)$, and value $V(i)$, with attention score $S(i, j) = Q(i)K(j)^\top$. We use strict future masking, so position i attends only to positions $j < i$. Soft attention applies softmax over the allowed scores, whereas masked hard attention selects a highest-scoring preceding position, breaking ties in favor of the rightmost one. B-RASP (Yang

et al., 2024) is a Boolean language for describing computations of masked hard-attention Transformers.

Stick-breaking attention (Tan et al., 2025) provides a differentiable relaxation of this rightmost-selection mechanism. For scores $S(i, j) \in [0, 1]$, $j < i$, it computes

$$\widetilde{S}(i, j) = S(i, j) \prod_{k=j+1}^{i-1} \big(1 - S(i, k)\big).$$

For binary scores, the rightmost preceding position with score 1 receives weight 1, and all other positions receive weight 0. If no such position exists, all weights are 0.

4 Learning Task

Our two transformer architectures are trained in a supervised manner on traces drawn from instances $P = \langle M, O, I, G\rangle$ of a STRIPS model $M = \langle \mathcal{P}, A\rangle$, containing ground actions with observed schema names and arguments over varying object sets O. We consider two supervision settings and extract STRIPS models for planning after training.

Under *applicability supervision*, a trace contains an initial state $s_0 = I$ and a sequence $\tau = (a_0, \ldots, a_{n-1})$ of applicable and inapplicable domain actions. Each action a_i receives label 0 if it is applicable in the state reached from s_0 by the preceding applicable actions, and 1 otherwise. We treat inapplicable actions as no-ops and mask them from subsequent attention. We use only the LS Transformer in this setting, since model extraction from the LB Transformer is not possible under this supervision, as explained later.

Under *final-state supervision*, a trace contains an initial state s_0, an applicable action sequence, and a set Q_F of ground-atom queries about the resulting state s_F. Each query $q \in Q_F$ receives label 0 if $q \in s_F$ and 1 otherwise. Both the LS and LB Transformers are trained in this setting. Neither setting reveals intermediate states.

We encode s_0 using an `init-start` action followed by an `init-`$p(\mathbf{o})$ action for each atom $p(\mathbf{o}) \in s_0$. The first initializes all ground atoms to false, while each `init-`$p(\mathbf{o})$ action sets $p(\mathbf{o})$ to true. A query $p(\mathbf{o}) \in Q_F$ is encoded as a `goal-`$p(\mathbf{o})$ action whose precondition is $p(\mathbf{o})$ and which has no effects. These actions encode predicate signatures and are shuffled independently within their groups.

In both settings, training minimizes focal loss (Lin et al., 2017), a variant of binary cross-entropy, over domain-action applicability labels in the first setting and final-state query labels in the second.

5 Transformer Architectures

5.1 Lifted STRIPS (LS) Transformer

To predict whether a domain action a_i is applicable, the LS Transformer checks whether all its ground preconditions hold. For each precondition $p(\mathbf{o})$, its truth value is determined by the most recent preceding domain or initialization action that affects it: an add effect makes it true, while a delete effect makes it false. The LS Transformer implements this computation through structured self-attention, extending the STRIPS Transformer of Núñez-Molina et al. (2026) to the lifted setting. It is based on the B-RASP program in Appendix B and described in detail in Appendix C.

The LS Transformer has a single self-attention layer with one head per hidden predicate $p \in \mathcal{P}'$. The number and arities of the hidden predicates are hyperparameters and need not match the ground-truth set $\mathcal{P}$. Each head checks preconditions involving its predicate.

For each predicate $p \in \mathcal{P}'$, action schema $\alpha \in A$, and valid pattern ρ, the parameters θ contain three scalars $\theta^{\text{pre}}_{p,\alpha,\rho}$, $\theta^{\text{eff}}_{p,\alpha,\rho}$, and $\theta^{\text{del}}_{p,\alpha,\rho}$. These are mapped to relaxed values $q_{p,\alpha,\rho}, k_{p,\alpha,\rho}, v_{p,\alpha,\rho} \in [0, 1]$, encoding whether the lifted atom $p(x_{\rho_1}, \ldots, x_{\rho_r})$ is a precondition, an effect, and a delete rather than an add effect, respectively. These relaxations enable end-to-end differentiable learning and are binarized for inference. After binarization, $q = 1$ specifies a precondition, while $k = 1$ specifies an effect whose sign is determined by v: $v = 0$ denotes an add effect and $v = 1$ a delete effect.

The auxiliary actions from Section 4 use fixed parameters: `init-`p and `goal-`p are tied to the same hidden predicate through the identity pattern, while `init-start` resets all atoms to false.

Each ground action $a_i = \alpha_i(\mathbf{o}_i)$ has a fixed binary embedding containing one-hot encodings of its schema and object arguments. Pattern-specific projections constructed from θ yield the scores

$$S_{p,\rho,\sigma}(i,j) = q_{p,\alpha_i,\rho}\, k_{p,\alpha_j,\sigma}\, \mathbb{I}\big[\mathbf{o}_i^{\rho} = \mathbf{o}_j^{\sigma}\big]\,, \tag{1}$$

where $\mathbf{o}_i^{\rho}$ is the object tuple selected by ρ. After binarization, the score is 1 exactly when (p,ρ) is a precondition of α_i, (p,σ) is an effect of α_j, and both patterns ground to the same atom.

Strict future masking restricts attention to $j < i$. For each precondition pattern ρ, stick-breaking normalization accounts for all effect patterns σ at every preceding position. After binarization, it selects the most recent effect on the corresponding ground atom, matching STRIPS semantics.

Combining the normalized weights with $V_{p,\sigma}(j) = v_{p,\alpha_j,\sigma}$ yields the violation score $Y_{p,\rho}(i)$. After binarization, this score is 1 exactly when (p,ρ) specifies a precondition of a_i and the corresponding ground atom is false. A continuous OR gives the action-level inapplicability score

$$y_\theta(i) = 1 - \prod_{p\in\mathcal{P}'} \prod_{\rho} \big(1 - Y_{p,\rho}(i)\big),$$

where the second product ranges over valid patterns. After binarization, $y_\theta(i) = 1$ iff at least one precondition is false, so a_i is applicable iff $y_\theta(i) = 0$.

The parameters θ are indexed by hidden predicates, action schemas, and patterns, but not by object identities or the number of objects. Changing the object set changes only the fixed action embeddings and projection matrices, allowing the same learned parameters to be instantiated on traces with different numbers of objects.

5.2 Lifted Stick-Breaking (LB) Transformer

The LB Transformer generalizes the SB Transformer of Núñez-Molina et al. (2026) to the lifted STRIPS setting. It is a decoder-style Transformer with stick-breaking attention (Tan et al., 2025), strict future masking, and no positional encodings. Unlike the LS Transformer, its parameters do not explicitly represent predicates, preconditions, or effects.

Each ground action $a_i = \alpha_i(o_{i,1}, \ldots, o_{i,k_i})$ is encoded as

$$\alpha_i,\ o_{i,1},\ \ldots,\ o_{i,k_i},\ \texttt{<sep>}.$$

Concatenating these sequences yields the trace representation. Action-schema and `<sep>` tokens use learned embeddings, while object tokens use randomized, non-learnable embeddings. Each action label is associated with its trailing `<sep>` token. A linear output layer applied to its final contextual representation yields $y_\theta(i) \in [0,1]$, the predicted probability that a_i is inapplicable given the preceding actions.

For each presentation of a trace, every object o is assigned a distinct embedding $r(o) \in \{-1,+1\}^{d_{\text{model}}}/\sqrt{d_{\text{model}}}$, sampled uniformly without replacement. The same embedding is reused for all occurrences of o within that trace, allowing the transformer to recognize when arguments of different actions refer to the same object. Object embeddings are resampled whenever a trace is sampled, encouraging the transformer to learn lifted operations that are invariant under object renamings. The representation supports up to $2^{d_{\text{model}}}$ distinct objects per trace. Because these embeddings are generated on demand rather than selected from a learned vocabulary, the same network can process traces with more objects than those seen during training.

6 Model Extraction and Planning

After training, we extract a lifted STRIPS model M' and discard the auxiliary `init-start`, `init-`p, and `goal-`p schemas. The extraction procedure depends on the supervision setting.

State reconstruction. Under final-state supervision, both Transformers are trained to answer ground-atom queries about the state reached by a trace. After training, we apply these queries

to action-sequence prefixes to infer unobserved intermediate-state values. For each prefix ending in a domain action a_i, we query all candidate atoms whose arguments are drawn from the objects in a_i and, when present, a_{i+1}, together with all nullary atoms. These are the only atoms that can be effects of a_i or preconditions of a_{i+1} under the STRIPS representation considered here. The observed initial state and these predictions provide the local pre- and post-state observations needed for every action occurrence.

For an occurrence $a_i = \alpha(\mathbf{o}_i)$, a queried atom $p(\mathbf{o}_i^\rho)$ is lifted to the candidate atom represented by (p, ρ). To tolerate prediction errors, we include a candidate as a precondition when it is predicted true before at least 95% of the occurrences of α. For effects, every occurrence votes for *add*, *delete*, or *none* according to its predicted transition, and we assign the modal label. A modal label of *none* yields no effect. The resulting preconditions and effects define M'. Although the LS Transformer also admits direct parameter extraction, we use this common architecture-independent procedure for both models under final-state supervision.

Direct extraction via symbolic alignment. Under applicability supervision, no successor-state queries are available, so state reconstruction cannot be used. For the LS Transformer, we instead extract M' directly from the binarized values $q_{p,\alpha,\rho}$, $k_{p,\alpha,\rho}$, and $v_{p,\alpha,\rho}$, which specify precondition membership, effect membership, and effect sign, respectively. The fixed `init-`p schemas align hidden predicate heads with the corresponding predicates p. Heads without such an alignment retain anonymous names. Because the LB parameters lack direct symbolic alignment, this extraction procedure does not apply to the LB Transformer, which we therefore do not evaluate under applicability supervision.

Planning. Given a problem $P = \langle M, O, I, G \rangle$, we replace M with the extracted model M' and translate the initial state and goal using the learned predicate alignment. We use Mimir (Ståhlberg, 2024) with greedy best-first search and the FF heuristic (Hoffmann & Nebel, 2001), limiting search to 10^7 generated nodes. We validate each returned plan against the ground-truth problem P under model M.

7 EXPERIMENTS

Our experiments address the following research questions:

- **RQ1:** How do the LS and LB Transformers compare against standard transformer baselines?
- **RQ2:** How does symbolic alignment affect performance under final-state supervision, and how does the supervision setting affect the LS Transformer?
- **RQ3:** Can the trained transformers yield explicit STRIPS models that support planning?
- **RQ4:** Do the learned models extrapolate to longer traces and larger object sets?

Baselines. We compare our models with two softmax-attention Transformer baselines: **Sinusoidal**, using sinusoidal positional encodings (Vaswani et al., 2017), and **RoPE**, using rotary position embeddings (Su et al., 2024), which can improve length generalization (Spies et al., 2025). Both use the same tokenization, randomized object embeddings, and network dimensions as LB. Positional information is applied to half of each token embedding to separate it partially from the randomized object representation.

Planning domains. We evaluate on standard automated-planning domains (Long & Fox, 2003; Fern et al., 2011; Taitler et al., 2024): **`blocksworld`** (block rearrangement), **`ferry`** (car transport by ferry), **`logistics`** (package delivery), **`npuzzle`** (sliding tiles), and **`maze`** (grid navigation).

Data generation. For each domain, we generate 100 problems for each of the training, test, and planning splits using randomized generators. Test and planning problems contain more objects than training problems. Training and test problems generate random traces whose numbers of domain actions are sampled uniformly from $[1, D]$. We generate 10^5 training traces with $D = 50$ and $2 \cdot 10^5$ test traces with $D = 200$. Exact object counts are reported in Appendix D.

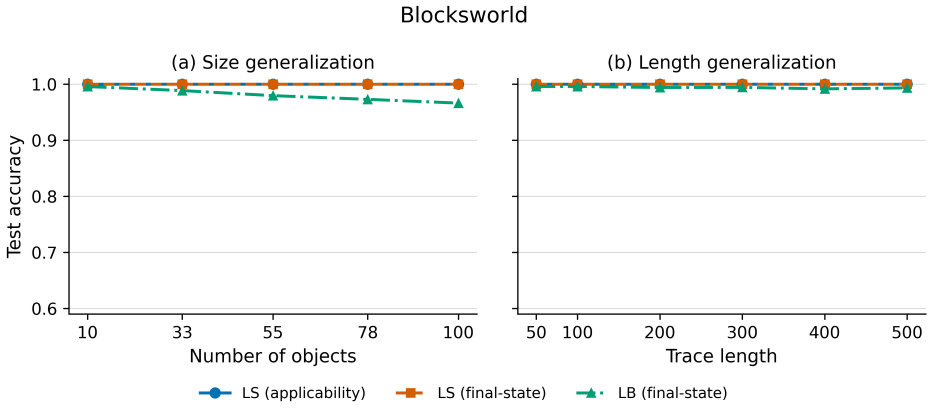


Figure 2: **Generalization in `blocksworld`.** Test accuracy for the LS and LB Transformers when varying **(a)** the number of objects up to $10\times$ the training maximum, with trace length fixed at $D = 50$, and **(b)** trace length up to $10\times$ the training maximum, with the number of objects fixed at its largest training value. For LS, we use the shared predicate configuration from Tables 1 and 2 and report both supervision settings. To assess generalization after successful fitting, we show only the seed with the highest training accuracy. Each point uses 10^4 traces, and the y-axis starts at 0.6. Results for the remaining domains are in Appendix F.

Under applicability supervision, each training action is sampled as applicable or inapplicable with equal probability. At test time, the prefix contains only applicable actions and only the final domain action may be inapplicable. Under final-state supervision, all domain actions are applicable, and each trace contains ten ground-atom queries sampled uniformly from all ground atoms over O.

Training and evaluation. The LS and LB Transformers are trained with RAdam (Liu et al., 2019). We use 10^5 training steps for LS and $5 \cdot 10^5$ for LB, increasing the latter to $7.5 \cdot 10^5$ in `logistics`. Every 10^4 steps, we compute accuracy on the full training set and retain the checkpoint with the highest training accuracy. We run each configuration with five random seeds and report mean performance, with the result of the seed achieving the highest training accuracy in parentheses.

Trace accuracy is the percentage of traces for which all supervised labels are predicted correctly. Under applicability supervision, these are the domain-action labels. Under final-state supervision, they are the final-state query labels. LS predictions use the binarized parameters $\bar{\theta}$, while LB predictions are thresholded at 0.5. For RQ4, we vary trace length and object count, keeping the other dimension at its largest training value. We evaluate 10^4 traces per setting and report the seed with the highest training accuracy. We extract STRIPS models using the procedure associated with each supervision setting. Planning accuracy is the percentage of planning problems for which the planner returns, within the search budget, a plan that is valid under the ground-truth model. All experiments run on an NVIDIA L40S GPU and an Intel Xeon Gold 6542Y CPU. Appendix E provides additional configuration details and model hyperparameters.

8 Results

RQ1: Comparison with standard Transformers. Tables 1 and 2 show that both softmax baselines perform poorly across domains, with low training and test accuracy and inaccurate extracted STRIPS models. Sinusoidal yields zero or near-zero planning accuracy in four of five domains, while RoPE does so in three. In contrast, LB achieves high training and test accuracy in every domain, solves all planning problems in three domains across all five seeds, and does so in `npuzzle` for the seed with the highest training accuracy.

We attribute this gap to LB's order-based recency bias from stick-breaking attention and strict future masking without positional encodings. This matches STRIPS semantics, where applicability depends on the most recent preceding action affecting each precondition, regardless of distance.

Table 1: **Main results.** Comparison of LS Transformer, LB Transformer and baseline models across five domains and two supervision settings, *final-state* and *applicability*, where the latter is only used by the LS Transformer. We report average training, test, and planning accuracy across five seeds, with the result of the seed achieving the highest training accuracy in parentheses. For the LS Transformer, we consider both the number of predicates in the ground-truth domain (*GT*) and a larger number of predicates shared across domains (*Shared*). Perfect values are highlighted in bold.

Blocksworld

	Final-state supervision			Applicability supervision		
Model	Train	Test	Plan	Train	Test	Plan
LS GT	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.439 (**1.0**)	.405 (**1.0**)	.238 (**1.0**)
LS Shared	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.606 (**1.0**)	.477 (**1.0**)	.602 (.020)
LB	.995 (.998)	.980 (.991)	**1.0** (**1.0**)	N/A	N/A	N/A
Sinusoidal	.186 (.186)	.457 (.458)	0.0 (0.0)	N/A	N/A	N/A
RoPE	.306 (.328)	.419 (.388)	0.0 (0.0)	N/A	N/A	N/A

Ferry

	Final-state supervision			Applicability supervision		
Model	Train	Test	Plan	Train	Test	Plan
LS GT	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.700 (**1.0**)	.627 (**1.0**)	.600 (**1.0**)
LS Shared	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.686 (**1.0**)	.618 (**1.0**)	.618 (**1.0**)
LB	.997 (.999)	.993 (.997)	**1.0** (**1.0**)	N/A	N/A	N/A
Sinusoidal	.410 (.847)	.502 (.635)	0.0 (0.0)	N/A	N/A	N/A
RoPE	.799 (.997)	.574 (.678)	.200 (**1.0**)	N/A	N/A	N/A

Logistics

	Final-state supervision			Applicability supervision		
Model	Train	Test	Plan	Train	Test	Plan
LS GT	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	1.000 (**1.0**)	1.000 (**1.0**)	.630 (**1.0**)
LS Shared	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.958 (**1.0**)	.769 (**1.0**)	.614 (**1.0**)
LB	.971 (.980)	.893 (.903)	.010 (.010)	N/A	N/A	N/A
Sinusoidal	.529 (.589)	.682 (.680)	.010 (.010)	N/A	N/A	N/A
RoPE	.972 (.978)	.401 (.398)	.010 (.010)	N/A	N/A	N/A

RQ2: Symbolic structure and supervision. Under final-state supervision, both architectures perform strongly, while LS achieves perfect accuracy across all reported metrics for every seed and domain. We attribute its advantage to the direct symbolic parameterization of preconditions and effects. Applicability-supervised LS shows high variance, but in every domain at least one predicate configuration achieves perfect accuracy across all metrics for its best seed. Thus, both supervision settings can recover the underlying model, although final-state supervision is easier to optimize. Successful runs with the Shared configuration also show that LS need not know the exact number of ground-truth predicates.

RQ3: STRIPS model extraction and planning. Appendix G reports the selected models, which recover the ground-truth model or an equivalent one in almost all cases. Two deviations deserve note. First, applicability-supervised LS learns extra `move` effects in `npuzzle` that preserve the dynamics but introduce irrelevant atoms (Listing 5). Mimir then exhausts its search budget on two problems, reducing planning accuracy to 0.98. Second, LB omits `at` preconditions from the `drive`, `fly`, and `load` actions in `logistics` (Listings 3–4), likely because approximately 95% of `at` atoms in the final states are false. LS recovers the correct `logistics` model under the same supervision. Except for the search-budget case above, correctly recovered models yield perfect planning, while their explicit form reveals the precise errors in incorrect models.

Table 2: **Continuation of Table 1.**

Npuzzle						
	Final-state supervision			Applicability supervision		
Model	Train	Test	Plan	Train	Test	Plan
LS GT	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.946 (.980)
LS Shared	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.808 (**1.0**)	.801 (**1.0**)	.740 (**1.0**)
LB	.884 (.992)	.889 (.989)	.600 (**1.0**)	N/A	N/A	N/A
Sinusoidal	.356 (.360)	.563 (.563)	0.0 (0.0)	N/A	N/A	N/A
RoPE	.386 (.395)	.510 (.470)	0.0 (0.0)	N/A	N/A	N/A

Maze						
	Final-state supervision			Applicability supervision		
Model	Train	Test	Plan	Train	Test	Plan
LS GT	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.808 (**1.0**)	.801 (**1.0**)	.800 (**1.0**)
LS Shared	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.615 (**1.0**)	.602 (**1.0**)	.600 (**1.0**)
LB	.981 (.989)	.970 (.981)	**1.0** (**1.0**)	N/A	N/A	N/A
Sinusoidal	.142 (.186)	.122 (.148)	**1.0** (**1.0**)	N/A	N/A	N/A
RoPE	.135 (.249)	.126 (.117)	.250 (**1.0**)	N/A	N/A	N/A

RQ4: Generalization to longer traces and more objects. Figure 2 reports results for `blocksworld`, with the remaining domains in Appendix F. We vary trace length and object count separately up to $10\times$ their training maxima. LS remains perfect in both dimensions under both supervision settings. LB also generalizes strongly, with its largest losses under object scaling: accuracy falls from 99.5% to 96.6% in `blocksworld` and from 95.6% to approximately 80% in `logistics`, where it plateaus for the largest instances.

9 Conclusion

Motivated by work on world models learned through next-token prediction, we studied whether Transformers can recover lifted STRIPS models from symbolic action traces. We introduced the LS Transformer, a symbolically-aligned architecture based on a B-RASP formulation of lifted STRIPS trace recognition, and the LB Transformer, a decoder-style Transformer with stick-breaking attention, no positional encodings, and randomized object embeddings. We compared final-state supervision, which provides queries about the resulting state, with applicability supervision, which labels actions without revealing successor states.

Under final-state supervision, LS achieves perfect training, test, and planning accuracy for every seed and domain, while LB attains high predictive accuracy, solves all planning problems in three domains across all five seeds, and does so in `npuzzle` for the seed with the highest training accuracy. Under applicability supervision, successful LS runs recover accurate models without successor states, although performance varies across seeds. For the best-fitting seeds, LS generalizes perfectly to traces up to ten times longer and, separately, to instances with up to ten times more objects. LB generalizes strongly, with larger losses under object scaling in some domains. The comparison with the softmax baselines suggests that recency-biased attention without positional encodings aligns with STRIPS semantics, while the consistency of LS shows the value of explicit symbolic alignment.

Most extracted models recover the ground-truth dynamics or equivalent ones and support off-the-shelf planning, while their explicit form reveals errors such as irrelevant effects and missing preconditions. Applicability supervision remains sensitive to initialization, and class imbalance can impair model extraction under final-state supervision. In future work, we plan to explore several extensions, including learning `STRIPS+` models (Jansen et al., 2025) where actions contain fewer arguments, learning from states represented as images (Xi et al., 2024; 2026), and learning stochastic world models.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks: **implementing methods**, including most of the LS and LB Transformer implementation, as well as scripts used to run experiments and aggregate results; **formulating mathematical claims**, specifically to identify an error in the B-RASP program of Appendix B and propose a correction, which was then verified step-by-step by the authors; **interpreting results**, specifically by identifying the class imbalance issue affecting predicate `at` in the `logistics` dataset. Additionally, we used generative AI tools to: **suggest experimental parameters**, by providing suitable problem generator parameters for scaling the number of objects in our generalization experiments; **improve phrasing and check grammar**; and **standardize mathematical notation**.

We have *not* used generative AI tools to: **generate synthetic data sets**, **help develop theoretical models or conceptual frameworks**, **propose or refine hypotheses**, **design or provide feedback on research methodology or experiments**, **assist with translation**, **clean and reformat dataset**, **support qualitative and thematic data analysis**. The following uses are not applicable: **proving mathematical claims** and **assisting in the writing of proofs**.

We have reviewed all AI-assisted work. All LLM-generated code was verified and tested by one of the authors. The correction to the B-RASP program in Appendix B was checked step-by-step for correctness. All LLM-assisted modifications to the main paper and appendices, including mathematical expressions, were reviewed by the authors.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide as supplementary material our anonymized source code, benchmark domain files, and scripts for data generation and aggregation of experimental results. Section 7 in the main paper describes the data-generation, training, and testing procedures, evaluation metrics, and hardware used. Appendix D describes the problem instances used to generate our datasets, whereas Appendix E provides additional information about the experimental configurations and model hyperparameters.

REFERENCES

Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 2019.

Abhiroop Ajith and Constantinos Chamzas. Strips-wm: Learning grounded propositional strips-style world models from images. *arXiv preprint arXiv:2606.06832*, 2026.

Masataro Asai, Hiroshi Kajino, Alex Fukunaga, and Christian Muise. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74:1599–1686, 2022.

Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained VLMs. *IEEE Robotics and Automation Letters*, 11(4):4002–4009, 2026. doi: 10.1109/LRA.2026.3662533.

Stephen N Cresswell, Thomas L McCluskey, and Margaret M West. Acquiring planning domain models using locm. *The Knowledge Engineering Review*, 28(2):195–213, 2013.

Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. *Machine Learning*, 84(1):81–107, 2011.

Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

Jonas Gösgens, Niklas Jansen, and Hector Geffner. Learning lifted STRIPS models from action traces alone: A simple, general, and scalable solution. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling*, volume 35, pp. 189–197, 2025. doi: 10.1609/icaps.v35i1.36117.

Jonas Gösgens, Niklas Jansen, and Hector Geffner. Learning lifted action models from traces with minimal information about actions and states. In *Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 929–939, 2026. doi: 10.24963/kr.2026/87.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 79081–79094. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f9f54762cbb4fe4dbffdd4f792c31221-Paper-Conference.pdf.

Yilun Hao, Yongchao Chen, Chuchu Fan, and Yang Zhang. Simulation to rules: A dual-VLM framework for formal visual planning. In *International Conference on Learning Representations*, 2026.

Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Jinbang Huang, Yixin Xiao, Zhanguang Zhang, Mark Coates, Jianye Hao, and Yingxue Zhang. One demo is all it takes: Planning domain derivation with LLMs from a single demonstration. In *International Conference on Learning Representations*, 2026.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *International Conference on Learning Representations*, volume 2025, pp. 58095–58179, 2025a.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=U49N5V51rU.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Niklas Jansen, Jonas Gösgens, and Hector Geffner. Learning lifted action models from traces of incomplete actions and states. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 832–842, 2025. doi: 10.24963/kr.2025/80.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.

Leonardo Lamanna, Luciano Serafini, Alessandro Saetti, Alfonso Emilio Gerevini, and Paolo Traverso. Lifted action models learning from partial traces. *Artificial Intelligence*, 339:104256, 2025.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, Joao F. Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations*, 2025.

Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B. Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. Exopredicator: Learning abstract models of dynamic worlds for robot planning. In *International Conference on Learning Representations*, 2026.

Songtuan Lin and Pascal Bercher. On the expressive power of planning formalisms in conjunction with ltl. *International Conference on Automated Planning and Scheduling*, 32(1):231–240, Jun. 2022.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision*, pp. 2980–2988, 2017.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30. Association for Computational Linguistics, December 2023. doi: 10.18653/v1/2023.blackboxnlp-1.2.

Carlos Núñez-Molina, Vicenç Gómez, and Hector Geffner. From next token prediction to (STRIPS) world models. In *Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 1075–1086, 2026. doi: 10.24963/kr.2026/101.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *34th International Conference on Automated Planning and Scheduling*, 2024. URL https://openreview.net/forum?id=C88wQIv0aJ.

Raanan Yehezkel Rohekar, Yaniv Gurwicz, Sungduk Yu, Estelle Aflalo, and Vasudev Lal. A causal world model underlying next token prediction: Exploring GPT in a controlled environment. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qA3xHJzF6B.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

Yash Sarrof, Yupei Du, Katharina Stein, Alexander Koller, Sylvie Thiébaux, and Michael Hahn. On the ability of transformers to verify plans. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026.

Alexander F Spies, William Edwards, Michael Ivanitskiy, Adrians Skapars, Tilman Räuker, Katsumi Inoue, Alessandra Russo, and Murray Shanahan. Transformers use causal world models in maze-solving tasks. In *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*, 2025.

Simon Ståhlberg. Mimir: A generalized planning library. https://github.com/simon-stahlberg/mimir, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, et al. The 2023 international planning competition, 2024.

Shawn Tan, Songlin Yang, Aaron Courville, Rameswar Panda, and Yikang Shen. Scaling stick-breaking attention: An efficient implementation and in-depth study. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 4030–4046, 2025.

Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. Chess as a testbed for language model state tracking. *AAAI Conference on Artificial Intelligence*, 36(10), 2022. doi: 10.1609/aaai.v36i10.21390. URL https://ojs.aaai.org/index.php/AAAI/article/view/21390.

Keyon Vafa, Justin Y Chen, Ashesh Rambachan, Jon Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37:26941–26975, 2024.

Keyon Vafa, Peter G. Chang, Ashesh Rambachan, and Sendhil Mullainathan. What has a foundation model found? inductive bias reveals world models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=i9npQatSev.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Kai Xi, Stephen Gould, and Sylvie Thiébaux. Neuro-symbolic learning of lifted action models from visual traces. In *International Conference on Automated Planning and Scheduling*, volume 34, pp. 653–662, 2024.

Kai Xi, Stephen Gould, and Sylvie Thiébaux. Learning lifted action models from unsupervised visual traces. In *Proceedings of the 36th International Conference on Automated Planning and Scheduling*, 2026.

Kaige Xie, Ian Yang, John Gunerli, and Mark Riedl. Making large language models into world models with precondition and effect knowledge. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 7532–7545. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.coling-main.503/.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *First Conference on Language Modeling*, 2024.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. *Advances in Neural Information Processing Systems*, 37:10202–10235, 2024.

Zhouliang Yu, Yuhuan Yuan, Tim Z. Xiao, Fuxiang Frank Xia, Jie Fu, Ge Zhang, Ge Lin, and Weiyang Liu. Generating symbolic world models via test-time scaling of large language models. *Transactions on Machine Learning Research*, 2025.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. In *International Conference on Learning Representations*, volume 2024, pp. 15898–15926, 2024.

Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence*, pp. 2444–2450. AAAI Press, 2013.

A THE B-RASP LANGUAGE

B-RASP is the Boolean counterpart of the RASP language (Weiss et al., 2021), designed to provide an abstract formalization of masked hard-attention transformers and simplify their theoretical analysis (Yang et al., 2024). In particular, a masked hard-attention transformer with some set of parameters can be translated into an equivalent B-RASP program, and vice versa. In this section, we provide an overview of the B-RASP language. Then, in next section, we will detail the B-RASP program for performing lifted STRIPS recognition.

A B-RASP program operates exclusively on Boolean vectors of length n. We index string and vector positions from 0 to $n-1$, and use $[n] = \{0, \ldots, n-1\}$. The input is a string

$$w = (w_0, \ldots, w_{n-1}),$$

whose symbols w_i belong to a finite alphabet Σ. This string is represented by a set of initial vectors I_σ, one for each symbol $\sigma \in \Sigma$, defined as

$$I_\sigma(i) = 1 \quad \text{iff} \quad w_i = \sigma, \qquad i \in [n].$$

Let the $|\Sigma|$ initial vectors I_σ be denoted by $P_1, \ldots, P_{|\Sigma|}$. A B-RASP program constructs additional vectors P_t from previously defined vectors $P_1, \ldots, P_{t-1}$ using two types of operations:

Position-wise operations. For every $i \in [n]$, the value $P_t(i)$ is given by a Boolean function of the values $\{P_1(i), \ldots, P_{t-1}(i)\}$ at the same position i.

Attention operations. For every $i \in [n]$, an attention operation has the form

$$P_t(i) := \blacktriangle_j \big[M(i,j), S(i,j)\big] V(i,j) : D(i), \qquad i \in [n], \tag{2}$$

where:

- $\blacktriangle_j$ is either $\blacktriangleleft_j$ (argmin) or $\blacktriangleright_j$ (argmax);
- the mask predicate $M(i,j)$ is either 1 (no masking), $j < i$ (strict future masking), or $j > i$ (strict past masking);
- the score predicate $S(i,j)$ and value predicate $V(i,j)$ are Boolean formulas, possibly different, over
$$\{P_1(i), \ldots, P_{t-1}(i)\} \cup \{P_1(j), \ldots, P_{t-1}(j)\};$$
- the default predicate $D(i)$ is a Boolean formula over
$$\{P_1(i), \ldots, P_{t-1}(i)\}.$$

For a given position $i \in [n]$, consider the indices $j \in [n]$ satisfying both $M(i,j) = 1$ and $S(i,j) = 1$. If at least one such index exists, let j_i be the smallest one when $\blacktriangle_j = \blacktriangleleft_j$, and the largest one when $\blacktriangle_j = \blacktriangleright_j$. The resulting value is then

$$P_t(i) = V(i, j_i).$$

If no position satisfies both predicates, the default value is used instead:

$$P_t(i) = D(i).$$

Intuitively, the operation can be interpreted as position i attending to a selected position j_i and returning the Boolean value $V(i, j_i)$. When no position j_i exists, the output is given by $D(i)$, which only depends on position i.

Multi-head attention can be represented by combining the results of several attention operations of the form in Eq. 2. This is particularly useful when compiling transformers into B-RASP, since independent computations corresponding to different attention heads at the same depth can be carried out in parallel.

B-RASP programs can be used either for language recognition or for sequence-to-sequence mappings. In the former case, the program determines whether an input string w belongs to a language $L \subseteq \Sigma^*$. After encoding w through the initial vectors above, the string is accepted iff

$$Z(n-1) = 1,$$

where Z is the final vector generated by the program.

Alternatively, a B-RASP program can map an input string $w \in \Sigma^*$ to an output string $w' \in \Gamma^*$ of the same length. In this setting, after encoding w as above, the output is represented by the last $|\Gamma|$ vectors Z_γ of the program, one for each $\gamma \in \Gamma$, with

$$w'(i) = \gamma \quad \text{iff} \quad Z_\gamma(i) = 1, \qquad i \in [n],\ \gamma \in \Gamma.$$

B B-RASP Program for Lifted STRIPS

We now provide a B-RASP Program for recognizing lifted STRIPS, which can be seen as a generalization of the one provided in (Núñez-Molina et al., 2026, Appendix B) for recognizing propositional STRIPS. In propositional STRIPS, actions have no variables and are instead grounded on a fixed set of objects O. Thus, the resulting propositional STRIPS model is tied to O and cannot generalize to a larger set of objects. We also take inspiration from (Sarrof et al., 2026, Appendix B), although we note that their program is encoded in C*-RASP[Pos], and ours in standard B-RASP.

Let $f_M(\tau) = 1$ iff some action is inapplicable when τ is evaluated from left to right under M, treating inapplicable actions as no-ops, and let $f_M(\tau) = 0$ otherwise. Our B-RASP program computes $f^{\text{B-RASP}}_{M,N}$, which agrees with f_M on traces involving at most N objects. We show that such a program can be constructed for any finite N. From the perspective of this program, the LS and LB Transformers can be viewed as learning $f^{\text{B-RASP}}_{M,N}$ for some number N of training objects, in a way that enables generalization to traces involving any larger number $N' > N$ of objects at test time.

Let $\mathcal{P}$ and A denote the finite sets of predicates and action schemas in M, respectively, and let $R = \max_{\alpha \in A} \operatorname{ar}(\alpha)$ be the maximum action arity. We consider an arbitrary set of N objects $O = \{o_1, \ldots, o_N\}$. A trace $\tau = (a_0, \ldots, a_{m-1})$ consists of ground actions

$$a_q = \alpha_q(o_{q,1}, \ldots, o_{q,\operatorname{ar}(\alpha_q)}), \qquad \alpha_q \in A,$$

and is encoded as the string

$$w_\tau = \alpha_0\, o_{0,1} \cdots o_{0,\operatorname{ar}(\alpha_0)} \ \cdots\ \alpha_{m-1}\, o_{m-1,1} \cdots o_{m-1,\operatorname{ar}(\alpha_{m-1})}.$$

Hence, for fixed M and N, the input alphabet $\Sigma_{M,N} = A \cup O$ is finite. Let n denote the length of w_τ, and let $i, j \in [n]$ denote positions in this string.

The input string w_τ is encoded by several initial vectors I_t, one for each $t \in \Sigma_{M,N}$, where

$$I_t(i) = 1 \quad \text{iff} \quad w_\tau(i) = t.$$

We first construct shifted versions of these vectors. Let $I^0_t = I_t$ and, for $k = 0, \ldots, R-1$, recursively define

$$I^{-(k+1)}_t(i) := \blacktriangleright_j\, [j < i, 1]\, I^{-k}_t(j) : 0. \tag{3}$$

Since $\blacktriangleright_j$ selects the largest $j < i$, $I^{-k}_t(i) = 1$ iff the token k positions before i is t, whenever such a position exists.

We perform the remaining computations at the last token of every ground action. For each action schema $\alpha \in A$, define

$$A_\alpha(i) := I^{-\operatorname{ar}(\alpha)}_\alpha(i). \tag{4}$$

Thus, $A_\alpha(i) = 1$ iff position i is the end of a ground instance of α.

For a predicate $p \in \mathcal{P}$ of arity $r = \operatorname{ar}(p)$, let

$$\mathcal{R}_p = \{1, \ldots, R\}^r$$

denote the set of possible argument patterns of p, with $\mathcal{R}_p = \{()\}$ when $r = 0$. If $\alpha(x_1, \ldots, x_k)$ is an action schema, an occurrence

$$p(x_{\rho_1}, \ldots, x_{\rho_r})$$

has pattern $\rho = (\rho_1, \ldots, \rho_r) \in \mathcal{R}_p$. Thus, a pattern maps predicate arguments to action arguments.

We denote by $\mathcal{R}^{\text{pre}}_{p,\alpha}$, $\mathcal{R}^{\text{add}}_{p,\alpha}$, and $\mathcal{R}^{\text{del}}_{p,\alpha}$ the sets of patterns with which p occurs in the preconditions, add effects, and delete effects of α, respectively, and define

$$\mathcal{R}^{\text{eff}}_{p,\alpha} = \mathcal{R}^{\text{add}}_{p,\alpha} \cup \mathcal{R}^{\text{del}}_{p,\alpha}.$$

These sets may contain several different patterns for the same predicate and action schema.

For each predicate p and pattern $\rho \in \mathcal{R}_p$, we define the Boolean vectors

$$\begin{aligned} Q_{p,\rho}(i) &:= \bigvee_{\substack{\alpha \in A| \\ \rho \in \mathcal{R}^{\text{pre}}_{p,\alpha}}} A_\alpha(i), \\ K_{p,\rho}(i) &:= \bigvee_{\substack{\alpha \in A| \\ \rho \in \mathcal{R}^{\text{eff}}_{p,\alpha}}} A_\alpha(i), \\ V_{p,\rho}(i) &:= \bigvee_{\substack{\alpha \in A| \\ \rho \in \mathcal{R}^{\text{del}}_{p,\alpha}}} A_\alpha(i). \end{aligned} \tag{5}$$

$Q_{p,\rho}(i)$ indicates that the action ending at i has p as a precondition with pattern ρ; $K_{p,\rho}(i)$ indicates that it has p as an effect with pattern ρ; and $V_{p,\rho}(i)$ indicates that this particular effect is a delete effect.

It remains to determine whether a precondition pattern of the action ending at i and an effect pattern of the action ending at j refer to the same ground atom. For every $p \in \mathcal{P}$, $\rho \in \mathcal{R}_p$, predicate argument $\ell \in \{1, \ldots, \text{ar}(p)\}$, and object $o \in O$, define

$$\begin{aligned} \text{Prec}_{p,\rho,\ell,o}(i) &:= \bigvee_{\substack{\alpha \in A| \\ \rho \in \mathcal{R}^{\text{pre}}_{p,\alpha}}} \left[A_\alpha(i) \wedge I_o^{-(\text{ar}(\alpha)-\rho_\ell)}(i) \right], \\ \text{Eff}_{p,\rho,\ell,o}(i) &:= \bigvee_{\substack{\alpha \in A| \\ \rho \in \mathcal{R}^{\text{eff}}_{p,\alpha}}} \left[A_\alpha(i) \wedge I_o^{-(\text{ar}(\alpha)-\rho_\ell)}(i) \right]. \end{aligned} \tag{6}$$

For example, if $\alpha(x, y, z)$ has precondition $p(z, x)$, its pattern is $\rho = (3, 1)$. At the end position of a ground action $\alpha(o_1, o_2, o_3)$, $\text{Prec}_{p,\rho,1,o_3}$ and $\text{Prec}_{p,\rho,2,o_1}$ are true.

For $\rho, \sigma \in \mathcal{R}_p$, we define the matching predicate

$$\chi_{p,\rho,\sigma}(i, j) := Q_{p,\rho}(i) \wedge K_{p,\sigma}(j) \wedge \bigwedge_{\ell=1}^{\text{ar}(p)} \bigvee_{o \in O} \left[\text{Prec}_{p,\rho,\ell,o}(i) \wedge \text{Eff}_{p,\sigma,\ell,o}(j) \right]. \tag{7}$$

Thus, $\chi_{p,\rho,\sigma}(i, j) = 1$ iff the action ending at i has p as a precondition with pattern ρ, the action ending at j has p as an effect with pattern σ, and both occurrences are instantiated on the same tuple of objects (i.e., they denote the same ground atom).

The predicates $Q_{p,\rho}$ and $K_{p,\sigma}$ are included explicitly in Eq. 7 to also cover predicates of arity zero. In this case, the conjunction over predicate arguments is empty and therefore evaluates to 1, so the existence of the corresponding precondition and effect must be checked through $Q_{p,()}$ and $K_{p,()}$. For predicates of arity larger than zero, these conditions are already implied by the Prec and Eff terms, but including them allows a single definition of χ for all predicate arities.

Importantly, χ is not an additional B-RASP operation. Equation 7 is simply shorthand for a Boolean formula over previously defined vectors at positions i and j, where j denotes the "attention index" of the corresponding attention operation (see Eq. 10 below). Also, since O is finite, the disjunction over $o \in O$ is finite and can therefore be expressed using standard B-RASP operations.

For each predicate p and precondition pattern $\rho \in \mathcal{R}_p$, define the score predicate

$$S_{p,\rho}(i, j) := \bigvee_{\sigma \in \mathcal{R}_p} \chi_{p,\rho,\sigma}(i, j), \tag{8}$$

and the value predicate

$$W_{p,\rho}(i, j) := \bigvee_{\sigma \in \mathcal{R}_p} \left[V_{p,\sigma}(j) \wedge \chi_{p,\rho,\sigma}(i, j) \right]. \tag{9}$$

The score predicate determines whether the action ending at j has some effect pattern σ that denotes the same ground atom as the precondition pattern ρ at i. Notice that $W_{p,\rho}(i,j)$ cannot in general be replaced by $V_{p,\rho}(j)$. While the score predicate identifies a matching position j, the effect pattern σ responsible for this match may be different from ρ. The value predicate therefore uses $\chi_{p,\rho,\sigma}$ to identify the matching effect pattern and returns its corresponding value $V_{p,\sigma}(j)$.

The applicability of the action ending at i with respect to the ground precondition represented by (p,ρ) is then computed by

$$Y_{p,\rho}(i) := \blacktriangleright_j \, [j < i, S_{p,\rho}(i,j)] \, W_{p,\rho}(i,j) : 0. \tag{10}$$

This operation attends to the most recent preceding action that affects the same ground atom as the precondition represented by (p,ρ). Its output is 1 if the corresponding matching effect is a delete effect, and 0 if it is an add effect. If no preceding action affects this ground atom, the default value is 0.

The attention operation in Eq. 10 is computed separately for every precondition pattern ρ. This is necessary when the same predicate occurs several times in the preconditions of an action: different ground occurrences of the predicate may have different most recent actions affecting them. Likewise, the value in Eq. 9 is associated with the matching effect pattern σ, allowing the same action schema to contain, for example, both an add and a delete effect of p under different patterns.

We then aggregate over all precondition patterns of the same predicate,

$$Y_p(i) := \bigvee_{\rho \in \mathcal{R}_p} Y_{p,\rho}(i), \tag{11}$$

and over all predicates,

$$Y(i) := \bigvee_{p \in \mathcal{P}} Y_p(i). \tag{12}$$

Hence, $Y(i) = 1$ iff the ground action ending at i is inapplicable given the preceding actions, meaning that at least one of its ground preconditions is false. At positions that do not end an action, $Y(i) = 0$.

Finally, the trace is negative iff some action in it is inapplicable. We compute

$$Z(i) := \blacktriangleright_j \, [1, Y(j)] \, 1 : 0. \tag{13}$$

Thus, $Z(i) = 1$ iff some action in the trace is inapplicable, and the output of the program is

$$f^{\text{B-RASP}}_{M,N}(\tau) = Z(n-1). \tag{14}$$

Theorem 1. *Let M be a lifted* STRIPS *model and let N be finite. For every action trace τ over M involving at most N objects,*

$$f^{\textit{B-RASP}}_{M,N}(\tau) = f_M(\tau).$$

The program above uses only standard B-RASP operations. For every finite N, the alphabet $\Sigma_{M,N}$ and all Boolean formulas in the program are finite. The program does, however, depend on N, as there is one initial vector I_o for every $o \in O$, and Eq. 7 ranges over these objects. Hence, the construction yields a B-RASP program $f^{\text{B-RASP}}_{M,N}$ for every finite, but arbitrarily large N, rather than a single B-RASP program over an unbounded object alphabet.

C Extended Description of LS Transformer

Here we present an extended description of Section 5.1 in the main paper, detailing the architecture of the LS Transformer. The LS Transformer can be viewed as the neural realization of the B-RASP program in Appendix B. Every set of parameter values θ defines a continuous relaxation of a (lifted) STRIPS model. Its binarization $\bar{\theta}$ directly specifies a discrete lifted STRIPS model. All operations in the LS Transformer are differentiable end-to-end. Hence, it provides a method for learning STRIPS models M with standard gradient-based optimization techniques, in a supervised manner from sequences of tokens encoding action traces over M.

Overview. The architecture exploits the structure of STRIPS action models directly. Consider a ground action a_i and one of its ground preconditions $p(\mathbf{o})$. Its truth value immediately before a_i is determined by the most recent preceding action that affects the same ground atom: $p(\mathbf{o})$ is true if this action adds it and false if it deletes it. Consequently, a_i is inapplicable iff at least one of its preconditions has a delete effect as its most recent preceding effect. The LS Transformer implements this computation directly. Each attention head represents a hidden predicate p; its query and key representations encode, respectively, possible occurrences of p in action preconditions and effects, while the values encode whether those effects are add or delete effects. Attention scores match preconditions with effects referring to the same ground atom, and stick-breaking attention retrieves the most recent matching effect. Finally, continuous OR operations aggregate the truth values of all ground preconditions to predict the applicability of each action in the trace.

Input representation. Let A be the set of action schemas and let $R = \max_{\alpha \in A} \mathrm{ar}(\alpha)$ denote the maximum action arity. Consider an input trace

$$\tau = (a_0, \ldots, a_{n-1}), \qquad a_i = \alpha_i(o_{i,1}, \ldots, o_{i,\mathrm{ar}(\alpha_i)}), \tag{15}$$

whose actions are grounded (i.e., instantiated) over a set $O = \{o_1, \ldots, o_N\}$ of N objects.

Each ground action a_i is represented by a single fixed binary embedding $e(a_i)$. We regard token embeddings and their projected representations as row vectors. The first $|A|$ entries form a one-hot encoding of its action schema α_i. These are followed by $|A|$ object blocks, each containing R one-hot vectors of dimension N. Each action a_i contains all-zero object blocks with the exception of its i-th block, which encodes its objects $o_{i,1}, \ldots, o_{i,\mathrm{ar}(\alpha_i)}$. For this block, the k-th one-hot vector identifies the object occurring as the k-th argument of a_i, while non-existing argument positions are represented by zero vectors. Thus,

$$e(a_i) \in \{0,1\}^{|A|+|A|RN}. \tag{16}$$

Although this representation depends on N, the transformer parameters introduced below do not.

Example of input representation. Assume $|A| = 3$, $N = 2$, and $R = 3$, and consider a ground action

$$a_i = \alpha_2(o_2, o_1), \qquad \mathrm{ar}(\alpha_2) = 2.$$

Its embedding is

$$e(a_i) = \underbrace{010}_{\text{action schema}} \; \underbrace{00\ 00\ 00}_{\alpha_1} \; \underbrace{01\ 10\ 00}_{\alpha_2} \; \underbrace{00\ 00\ 00}_{\alpha_3}.$$

The first three bits encode the action schema α_2. The remaining bits are divided into one object block per action schema. Since a_i is an instance of α_2, the blocks associated with α_1 and α_3 are all zero. Within the α_2 block, the first one-hot vector 01 encodes o_2, the second vector 10 encodes o_1, and the third vector 00 corresponds to the unused third argument position, since $\mathrm{ar}(\alpha_2) = 2 < R$.

Lifted patterns and model parameters. To simplify notation, in this appendix $\mathcal{P}$ denotes the hidden-predicate set $\mathcal{P}'$ from Section 5.1. Each $p \in \mathcal{P}$ has a fixed arity $r = \mathrm{ar}(p)$ and is associated with one attention head. A *pattern*

$$\rho = (\rho_1, \ldots, \rho_r) \in \{1, \ldots, R\}^r \tag{17}$$

specifies how an occurrence of p is instantiated on the arguments of an action schema, i.e., it maps predicate arguments to action arguments. For example, if $\alpha(x_1, x_2, x_3)$ is an action schema and p is binary, the pattern $\rho = (3, 1)$ represents the lifted atom $p(x_3, x_1)$. For predicates of arity zero, we use the single empty pattern $\rho = ()$.

Not every pattern is valid for every action schema. For an action schema α, we define

$$\mu_{\alpha,\rho} = \begin{cases} 1, & \text{if } \rho_1, \ldots, \rho_r \text{ are pairwise distinct and } \rho_\ell \le \mathrm{ar}(\alpha) \text{ for all } \ell, \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

Hence, we mask patterns referring to non-existing action arguments and those with repeated indices, thus disallowing lifted atoms with repeated action arguments such as $p(x_1, x_1)$. For $r = 0$, the empty pattern is always valid.

We denote the complete set of LS Transformer parameters by θ. For every predicate p, action schema α, and pattern ρ, the transformer contains three scalar parameters

$$\theta^{\text{pre}}_{p,\alpha,\rho}, \qquad \theta^{\text{eff}}_{p,\alpha,\rho}, \qquad \theta^{\text{del}}_{p,\alpha,\rho}. \tag{19}$$

Once they are mapped to values in $[0, 1]$, they represent, respectively, the degree to which p with pattern ρ is a precondition of α, the degree to which it is an effect of α, and the degree to which it is a delete rather than an add effect. Different pattern slots are independent. An action may therefore contain several occurrences of the same predicate with different patterns, such as both $p(x_1, x_2)$ and $p(x_2, x_1)$.

We denote the corresponding probabilities by

$$\begin{aligned} q_{p,\alpha,\rho} &= \mu_{\alpha,\rho}\, h_T\big(\theta^{\text{pre}}_{p,\alpha,\rho}\big)\,, \\ k_{p,\alpha,\rho} &= \mu_{\alpha,\rho}\, h_T\big(\theta^{\text{eff}}_{p,\alpha,\rho}\big)\,, \\ v_{p,\alpha,\rho} &= \mu_{\alpha,\rho}\, h_T\big(\theta^{\text{del}}_{p,\alpha,\rho}\big)\,. \end{aligned} \tag{20}$$

In our implementation, these parameters are stored as logits. At inference time, we first map each logit $w \in \theta$ to a probability using

$$h_T(w) = \sigma(w), \tag{21}$$

and then discretize the resulting probability to $\{0, 1\}$ using a threshold of 0.5.

During training, we instead use the Gumbel-sigmoid relaxation (Jang et al., 2017)

$$h_T(w) = \sigma\left(\frac{w + g_1 - g_2}{T}\right), \qquad g_1, g_2 \sim \text{Gumbel}(0, 1), \tag{22}$$

where $T > 0$ is the temperature, which is annealed during training. This provides a differentiable approximation to the binary decisions represented by the STRIPS parameters θ.

Query, key, and value projections. As in a standard transformer, each predicate $p \in \mathcal{P}$ defines query, key, and value projections that map the input embedding $e(a_i)$ into the representations used by its attention head. These projections are highly structured: rather than being arbitrary learned matrices, they encode the action argument permutations given by the patterns ρ, weighted by the learned precondition, effect, and effect-sign values of Eq. 20.

Consider first a predicate p of arity $r > 0$. For a ground action a_i, let $\mathbf{u}_{i,k} \in \{0, 1\}^N$ denote the one-hot vector identifying the object occurring at its k-th argument position, and define

$$\mathbf{u}_i = [\mathbf{u}_{i,1} \quad \cdots \quad \mathbf{u}_{i,R}] \in \{0, 1\}^{RN}, \tag{23}$$

where non-existing argument positions are represented by zero vectors.

For each pattern $\rho = (\rho_1, \ldots, \rho_r)$, let

$$P^N_\rho \in \{0, 1\}^{RN \times rN} \tag{24}$$

be a fixed pattern-selection matrix. Multiplication by P^N_ρ selects and orders the r action arguments specified by ρ:

$$\mathbf{u}_i P^N_\rho = [\mathbf{u}_{i,\rho_1} \quad \cdots \quad \mathbf{u}_{i,\rho_r}]\,. \tag{25}$$

Hence, P^N_ρ depends only on the pattern ρ and the number of objects N, and contains no learnable parameters.

We use these selection matrices to construct the query and key projections. Let $A = \{\alpha_1, \ldots, \alpha_{|A|}\}$. For each predicate p and pattern ρ, the corresponding query projection matrix is

$$W^{p,Q}_\rho = \begin{bmatrix} \mathbf{0}_{|A| \times rN} \\ q^{1/r}_{p,\alpha_1,\rho} P^N_\rho \\ q^{1/r}_{p,\alpha_2,\rho} P^N_\rho \\ \vdots \\ q^{1/r}_{p,\alpha_{|A|},\rho} P^N_\rho \end{bmatrix} \in \mathbb{R}^{(|A|+|A|RN) \times rN}, \tag{26}$$

while the key projection matrix is

$$W_\rho^{p,K} = \begin{bmatrix} \mathbf{0}_{|A|\times rN} \\ k_{p,\alpha_1,\rho}^{1/r} P_\rho^N \\ k_{p,\alpha_2,\rho}^{1/r} P_\rho^N \\ \vdots \\ k_{p,\alpha_{|A|},\rho}^{1/r} P_\rho^N \end{bmatrix} \in \mathbb{R}^{(|A|+|A|RN)\times rN}. \tag{27}$$

The first $|A|$ rows correspond to the action-schema one-hot part of $e(a_i)$ and are zero, while each subsequent block corresponds to the object block of one action schema. Thus, the query and key projection matrices contain one copy of P_ρ^N for every action schema α_j, weighted by the r-th root of the corresponding precondition probability $q_{p,\alpha_j,\rho}$ or effect probability $k_{p,\alpha_j,\rho}$. The exponent $1/r$ normalizes across predicate arguments, so that multiplying the r argument-wise matching scores recovers the original precondition and effect probabilities.

Since $e(a_i)$ has a non-zero object block only for its action schema α_i, multiplying it by the pattern-specific projection matrices $W_\rho^{p,Q}$ and $W_\rho^{p,K}$ yields the corresponding query and key vectors

$$\mathbf{Q}_{p,\rho}(i) = e(a_i)W_\rho^{p,Q} = q_{p,\alpha_i,\rho}^{1/r}\, \mathbf{u}_i P_\rho^N, \tag{28}$$

and

$$\mathbf{K}_{p,\rho}(i) = e(a_i)W_\rho^{p,K} = k_{p,\alpha_i,\rho}^{1/r}\, \mathbf{u}_i P_\rho^N. \tag{29}$$

Stacking $W_\rho^{p,Q}$ and $W_\rho^{p,K}$ for all patterns ρ of p yields the query and key projection tensors $W^{p,Q}$ and $W^{p,K}$, respectively.

For the remainder of the description, it is convenient to split each rN-dimensional projected representation into r vectors of dimension N. Using Eq. 25, for $\ell \in \{1, \ldots, r\}$ we write

$$\mathbf{Q}_{p,\rho,\ell}(i) = q_{p,\alpha_i,\rho}^{1/r}\, \mathbf{u}_{i,\rho_\ell}, \tag{30}$$

and

$$\mathbf{K}_{p,\rho,\ell}(i) = k_{p,\alpha_i,\rho}^{1/r}\, \mathbf{u}_{i,\rho_\ell}. \tag{31}$$

The value projection does not need to encode object identities. For each predicate p and pattern ρ, we define the matrix

$$W_\rho^{p,V} = \begin{bmatrix} v_{p,\alpha_1,\rho} \\ v_{p,\alpha_2,\rho} \\ \vdots \\ v_{p,\alpha_{|A|},\rho} \\ \mathbf{0}_{|A|RN\times 1} \end{bmatrix} \in \mathbb{R}^{(|A|+|A|RN)\times 1}. \tag{32}$$

The first $|A|$ entries correspond to the action-schema one-hot part of $e(a_i)$, while all entries associated with the object blocks are zero. The corresponding scalar value is obtained as

$$V_{p,\rho}(i) = e(a_i)W_\rho^{p,V} = v_{p,\alpha_i,\rho}. \tag{33}$$

Thus, $V_{p,\rho}(i)$ encodes the effect sign of pattern ρ of p for action schema α_i. The vectors $W_\rho^{p,V}$ can analogously be stacked into a single value projection matrix $W^{p,V}$.

For an arity-zero predicate, there are no object arguments or pattern-selection matrices. In this case, the action a_i is projected to the following scalar query, key and value:

$$\begin{aligned} Q_{p,()}(i) &= q_{p,\alpha_i,()}, \\ K_{p,()}(i) &= k_{p,\alpha_i,()}, \\ V_{p,()}(i) &= v_{p,\alpha_i,()}. \end{aligned} \tag{34}$$

Score computation. Consider a query pattern ρ of the action a_i and an effect pattern σ of an action a_j. For a predicate p of arity $r > 0$, its attention head computes the score

$$S_{p,\rho,\sigma}(i,j) = \prod_{\ell=1}^{r} \langle \mathbf{Q}_{p,\rho,\ell}(i), \mathbf{K}_{p,\sigma,\ell}(j) \rangle \,. \tag{35}$$

For a predicate of arity zero, the score is simply

$$S_{p,(),()}(i,j) = Q_{p,()}(i)\, K_{p,()}(j). \tag{36}$$

Because object arguments are one-hot encoded, the score in Eq. 35 is zero whenever the patterns ρ, σ induce different ground atoms. If they induce the same ground atom, all corresponding object vectors match and

$$S_{p,\rho,\sigma}(i,j) = q_{p,\alpha_i,\rho}\, k_{p,\alpha_j,\sigma}. \tag{37}$$

Hence, the score measures whether a_i contains a ground instance of p as a precondition and a_j affects that same ground instance.

Causal masking and stick-breaking attention. The applicability of a_i can only depend on preceding actions. We therefore apply strict future masking,

$$S_{p,\rho,\sigma}(i,j) = 0 \qquad \text{for } j \geq i. \tag{38}$$

In traces containing inapplicable actions at intermediate positions, such actions are additionally masked as keys for all subsequent positions, making the applicability of a_i independent of previous inapplicable actions. This is equivalent to considering that inapplicable actions do not modify the state.

Given predicate p, query pattern ρ, and positions i and j, we first aggregate the scores over the effect patterns of a_j:

$$\bar{S}_{p,\rho}(i,j) = \sum_{\sigma} S_{p,\rho,\sigma}(i,j). \tag{39}$$

At most one effect pattern σ can yield a non-zero score for a given (p, ρ, i, j), since two distinct valid patterns of the same action cannot instantiate the same ground atom. Hence, $\bar{S}_{p,\rho}(i,j)$ is simply the score of the unique matching effect pattern, if it exists.

Then, scores are normalized using stick-breaking attention (Tan et al., 2025):

$$\widetilde{S}_{p,\rho,\sigma}(i,j) = S_{p,\rho,\sigma}(i,j) \prod_{k=j+1}^{i-1} \left(1 - \bar{S}_{p,\rho}(i,k)\right). \tag{40}$$

This operation induces a recency bias that aligns with STRIPS semantics, since the truth value of a ground precondition depends only on the most recent preceding action that affects it. The operation is applied independently for every query pattern ρ, so different ground preconditions involving the same predicate p may attend to different preceding effects.

Value aggregation and action applicability. For each predicate p and precondition pattern ρ, the normalized scores are combined with the effect values according to

$$Y_{p,\rho}(i) = \sum_{j<i} \sum_{\sigma} \widetilde{S}_{p,\rho,\sigma}(i,j) V_{p,\sigma}(j). \tag{41}$$

$Y_{p,\rho}(i)$ represents the degree to which the ground precondition given by (p, ρ) is false immediately before a_i. For binarized parameters, if its most recent matching effect is a delete effect, then $Y_{p,\rho}(i) = 1$; if it is an add effect, then $Y_{p,\rho}(i) = 0$. If no preceding action affects the ground atom, all corresponding attention scores are zero and hence $Y_{p,\rho}(i) = 0$. Therefore, preconditions are true by default.

Next, ground preconditions corresponding to different patterns ρ of the same predicate p are combined using a continuous OR operation

$$Y_p(i) = 1 - \prod_{\rho} \left(1 - Y_{p,\rho}(i)\right). \tag{42}$$

Finally, the outputs of all predicate heads are combined in the same manner:

$$y_\theta(i) = 1 - \prod_{p \in \mathcal{P}} \left(1 - Y_p(i)\right). \tag{43}$$

The LS Transformer therefore returns one prediction

$$y_\theta(i) \in [0, 1] \tag{44}$$

for every action a_i in the trace, where 0 represents applicable and 1 represents inapplicable.

Encoding initial states and goals. Information about initial and final states can be represented using the setup actions introduced in the main paper. These actions use fixed rather than learned parameters, and connect predicates of the ground-truth model with the corresponding hidden predicates of the transformer.

For a predicate p of arity r, let

$$\iota_p = (1, \ldots, r) \tag{45}$$

denote its identity pattern, with $\iota_p = ()$ when $r = 0$. Action schemas of the form $\texttt{init-}p(x_1, \ldots, x_r)$ are encoded as having no preconditions and a single add effect $p(x_1, \ldots, x_r)$, given by the pattern ι_p. Conversely, action schemas of the form $\texttt{goal-}p(x_1, \ldots, x_r)$ have $p(x_1, \ldots, x_r)$ as single precondition and no effects. Thus, their predicted applicability acts as a query on the value of the lifted atom $p(x_1, \ldots, x_r)$ at the corresponding point in the trace. Also, $\texttt{init-}p$ and $\texttt{goal-}p$ actions referring to the same ground-truth predicate p are tied to the same hidden predicate.

The special action `init-start`, placed before all $\texttt{init-}p$ actions, initializes all ground atoms to false. Its embedding contains ones at every object position, and it is encoded as having one delete effect for every hidden predicate. After `init-start`, the $\texttt{init-}p$ actions will add the atoms that are true in the initial state, yielding a closed-world representation where the value of every atom is known.

Binarization and symbolic interpretation. After training, parameters θ are converted to probabilities using Eq. 21 and then discretized using a threshold of 0.5, resulting in a set of binarized parameters $\bar{\theta}$. For every valid triple (p, α, ρ), a binarized precondition parameter equal to one means that p under pattern ρ is a precondition of α, while a binarized effect parameter equal to one means that it is an effect of α. For an active effect, its sign parameter determines whether it is an add or delete effect. Thus, the binarized parameters $\bar{\theta}$ directly specify a STRIPS model.

With binary parameters, object matching in Eq. 35 becomes exact, stick-breaking attention selects the most recent preceding effect on each ground precondition, and Eqs. 41–43 compute action applicability exactly. Consequently, the binarized LS Transformer implements the per-action computation underlying the B-RASP program in Appendix B. For traces containing intermediate inapplicable actions, it additionally masks those actions from later positions, implementing the no-op convention of Section 4.

Generalization over objects. The parameters θ of the LS Transformer depend only on the number of hidden predicates and action schemas, along with their arities. In particular, they do not depend on the number N of objects or their identities. The value of N only affects action embeddings and how transformer parameters are instantiated on the projection matrices in Eqs. 26, 27, and 32. The same learned parameters can therefore be instantiated on traces with any finite number of objects. Their generalization to larger object sets is evaluated empirically in Section 8.

D Problem Instances

This appendix describes the problem instances used in our experiments. We differentiate between problems used for training, problems used for testing and planning in the main experiments, and problems used to evaluate generalization to larger numbers of objects. For every set of problems, we detail both the total number N of objects and objects of each particular type, providing ranges for each.

D.1 Training, test, and planning problems

For each domain, we generate 100 training problems, 100 test problems, and 100 planning problems using different random seeds. Table 3 gives their sizes. Test and planning problems use the same size ranges.

Table 3: Problem sizes for the main experiments. Each domain has 100 problems in each of the training, test, and planning splits. Integer intervals are inclusive; entries containing several intervals specify their Cartesian product.

	Training		Test / planning	
Domain	Size parameters	N	Size parameters	N
`blocksworld`	$b \in [4, 10]$	4–10	$b \in [11, 20]$	11–20
`ferry`	$(c, p) \in [2, 5]^2$	4–10	$(c, p) \in [6, 10]^2$	12–20
`logistics`	$(c, p, t, a) \in [1, 2]^4$ $\ell \in [1, 3]$	6–13	$(c, p, t, a) \in [3, 4]^4$ $\ell \in [4, 5]$	19–25
`npuzzle`	$2\times3,\ 3\times3$	11, 17	$3\times4,\ 4\times4$	23, 31
`maze`	$2\times3,\ 3\times3$	6, 9	$(r, c) \in [4, 5]^2$	16, 20, 25

In `blocksworld`, b is the number of blocks and $N = b$. In `ferry`, c and p count cars and ports, respectively, giving $N = c + p$. In `logistics`, (c, p, t, a, ℓ) counts cities, packages, trucks, airplanes, and non-airport locations. The generator adds one airport per city, so $N = 2c+p+t+a+\ell$. Here, ℓ counts non-airport locations across all cities. For an $r \times c$ grid, `npuzzle` has rc positions and $rc - 1$ tiles, giving $N = 2rc - 1$, whereas `maze` has $N = rc$ cells, including those that are blocked.

D.2 Problems for Size Generalization

We generate 100 additional problems at each of five fixed sizes per domain, using different seeds from the previous datasets. Table 4 lists their sizes and object counts. The five target sizes are approximately evenly spaced between the maximum number of objects during training and ten times that value. The table reports the actual configurations and object counts.

Table 4: Problem sizes for generalization experiments, with 100 problems per size and domain. For `logistics`, $k = c = p = t = a$ and ℓ counts non-airport locations. Column *Size 1* contains the problem sizes used for the length generalization experiments.

Domain	Type	Size 1	Size 2	Size 3	Size 4	Size 5
`blocksworld`	$b = N$	10	33	55	78	100
`ferry`	$c = p$	5	16	28	39	50
	N	10	32	56	78	100
`logistics`	(k, ℓ)	$(2, 3)$	$(7, 10)$	$(11, 17)$	$(16, 23)$	$(20, 30)$
	N	13	45	72	103	130
`npuzzle`	Grid	3×3	5×6	6×8	8×8	8×11
	N	17	59	95	127	175
`maze`	Grid	3×3	5×6	7×7	7×10	9×10
	N	9	30	49	70	90

E Experimental Configuration and Model Hyperparameters

Table 5 reports training settings and model hyperparameters. These settings are shared across domains and supervision settings, with the exception of the predicate counts and training budgets indicated below. We use seeds 1–5 for training. Every 10,000 training steps, we evaluate trace accuracy on the full training dataset and retain the best checkpoint.

Table 5: Model and optimization settings. The sinusoidal and RoPE baselines share the LB configuration, with the attention and positional-encoding changes described below.

Setting	LS	LB / baselines
Transformer layers	1	8
Attention heads per layer	One per hidden predicate	8
Embedding dimension d_{model}	N/A	256
Feed-forward dimension	N/A	1024
Feed-forward activation	N/A	GELU
Normalization	N/A	Pre-LayerNorm
Dropout / attention dropout	0 / 0	0.1 / 0.1
Optimizer	RAdam	RAdam
Learning rate	10^{-2}	5×10^{-5}
Batch size (traces)	16	16
Training steps	100,000	500,000
Training steps in `logistics`	100,000	750,000
Focal-loss exponent	1	1
`bce-weight` for domain actions	0.999	0.9
`bce-weight` for `goal-`p actions	0.5	0.5
LS parameter representation	Logits	N/A
LS initialization method	z	N/A
Gumbel temperature	$1 \to 0.2$ in 50,000 steps	N/A

Focal loss weighting. In the focal loss used for training, terms with labels 0 and 1 are weighted differently. The parameter `bce-weight` denotes the label-1 weight for domain actions, with its complement being used for label-0 terms. Auxiliary `goal-`p actions use a weight of 0.5 for both labels.

Table 6: LS hidden-predicate configurations, used under both supervision settings. (P_0, P_1, P_2) gives the number of predicates of arity 0, 1, and 2; their sum gives the number of attention heads.

Configuration	(P_0, P_1, P_2)	Total heads
GT: `blocksworld`	$(1, 3, 1)$	5
GT: `ferry`	$(1, 4, 1)$	6
GT: `logistics`	$(0, 7, 3)$	10
GT: `npuzzle`	$(0, 3, 2)$	5
GT: `maze`	$(0, 2, 1)$	3
Shared: all domains	$(1, 10, 5)$	16

LS initialization and relaxation. The parameters of the LS Transformer are initialized uniformly in the $[0, 0.1)$ range for those encoding precondition and effect probabilities (θ^{pre} and θ^{eff}), and in $[0, 1)$ for the effect-sign probabilities (θ^{del}). After initialization, they are converted from probabilities to logits, and invalid pattern slots are masked. Parameters encoding auxiliary actions are fixed: `init-start` has no preconditions and sets every atom false; `init-p` has its associated atom as its single add effect, and no preconditions; and `goal-p` has its associated atom as its single precondition, and no effects. The Gumbel-sigmoid temperature decreases linearly from 1 to 0.2 over the first 50,000 steps and remains at 0.2 afterwards. Evaluation disables Gumbel-sigmoid and binarizes the probabilities at 0.5.

LB and baseline representations. Action schemas and the separator token have learned embeddings. Object embeddings are non-learnable vectors in $\{-1, +1\}^{d_{\text{model}}}/\sqrt{d_{\text{model}}}$, resampled on every forward pass, with one vector per object within a trace and ensuring two different objects do not get the same vector. Predictions are read at the separator following each action. LB uses no positional encodings. The baselines use softmax attention with either sinusoidal encodings on the last half of the embedding coordinates, scaled by $\sqrt{2/d_{\text{model}}}$, or RoPE on the last half of each query/key head, with base 10,000.

F Extended Generalization Results

In this section we extend Figure 2 in the main paper by providing generalization results for the remaining four domains: `ferry`, `logistics`, `maze` and `npuzzle`.

G Extracted STRIPS Models

We report the complete STRIPS models extracted in every domain from the seed with the highest training accuracy for each architecture and supervision setting. For LS, we use the GT predicate configuration.

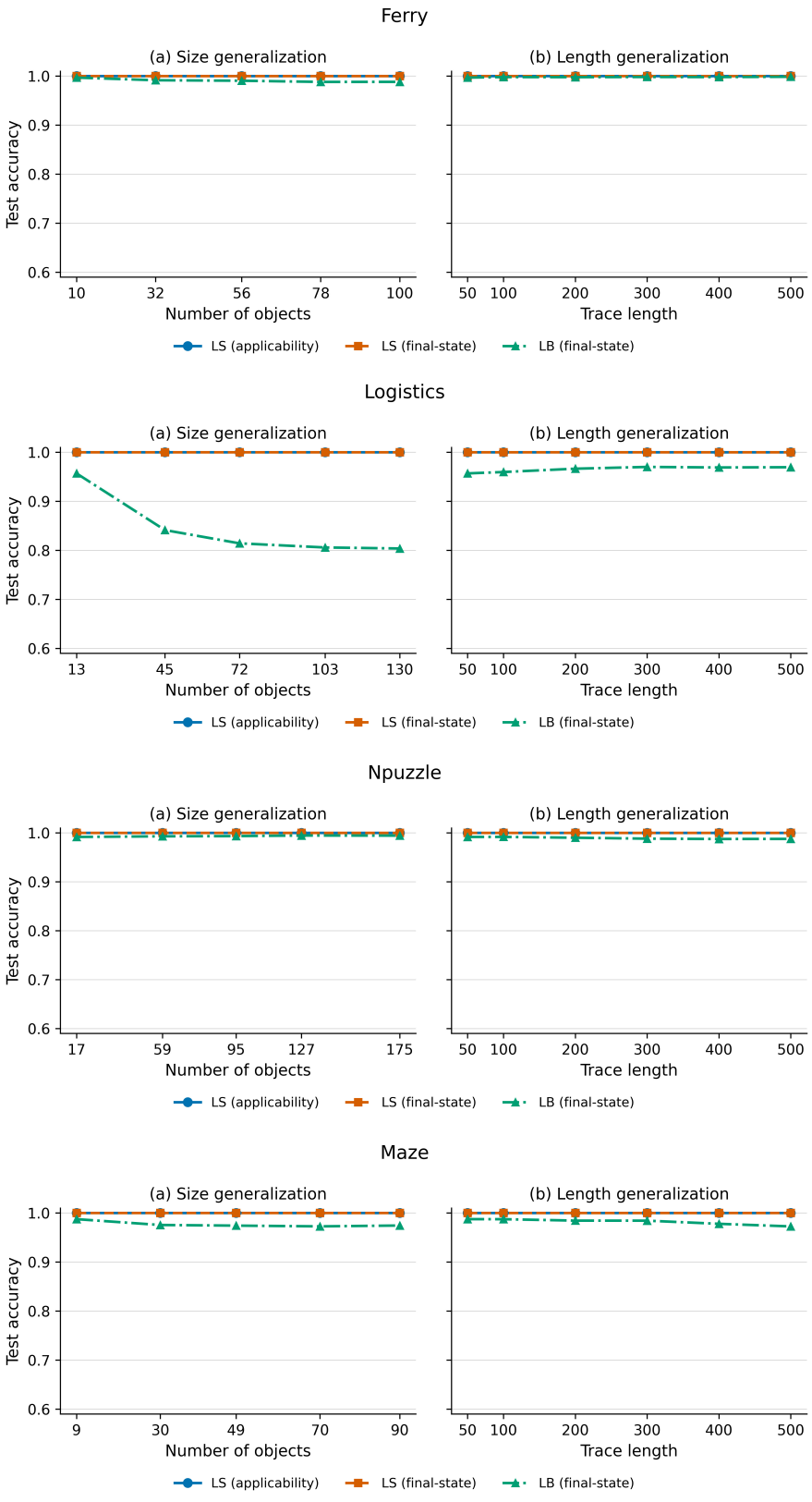


Figure 3: **Generalization in `ferry`, `logistics`, `maze`, and `npuzzle`.**

(a) LS (applicability)

```
(:predicates
 (clear ?x)
 (handempty)
 (on ?x ?y)
 (ontable ?x)
 (p3 ?x))

(:action pick-up
 :parameters (?x)
 :precondition (and
  (clear ?x)
  (handempty)
  (ontable ?x))
 :effect (and
  (clear ?x)
  (ontable ?x)
  (p3 ?x)
  (not (handempty))))

(:action put-down
 :parameters (?x)
 :precondition (and
  (clear ?x)
  (ontable ?x)
  (p3 ?x))
 :effect (and
  (handempty)
  (not (p3 ?x))))

(:action stack
 :parameters (?x ?y)
 :precondition (and
  (clear ?y)
  (clear ?x)
  (ontable ?x)
  (p3 ?x))
 :effect (and
  (handempty)
  (on ?x ?y)
  (not (clear ?y))
  (not (ontable ?x))
  (not (p3 ?x))))

(:action unstack
 :parameters (?x ?y)
 :precondition (and
  (clear ?x)
  (handempty)
  (on ?x ?y))
 :effect (and
  (clear ?y)
  (clear ?x)
  (ontable ?x)
  (p3 ?x)
  (not (handempty))
  (not (on ?x ?y))))
```

(b) LS (final-state)

```
(:predicates
 (clear ?x)
 (handempty)
 (on ?x ?y)
 (ontable ?x)
 (holding ?x))

(:action pick-up
 :parameters (?x)
 :precondition (and
  (clear ?x)
  (handempty)
  (ontable ?x))
 :effect (and
  (holding ?x)
  (not (handempty))
  (not (clear ?x))
  (not (ontable ?x))))

(:action put-down
 :parameters (?x)
 :precondition (and
  (holding ?x))
 :effect (and
  (handempty)
  (clear ?x)
  (ontable ?x)
  (not (holding ?x))))

(:action stack
 :parameters (?x ?y)
 :precondition (and
  (clear ?y)
  (holding ?x))
 :effect (and
  (handempty)
  (on ?x ?y)
  (clear ?x)
  (not (clear ?y))
  (not (holding ?x))))

(:action unstack
 :parameters (?x ?y)
 :precondition (and
  (clear ?x)
  (handempty)
  (on ?x ?y))
 :effect (and
  (clear ?y)
  (holding ?x)
  (not (handempty))
  (not (on ?x ?y))
  (not (clear ?x))))
```

(c) LB (final-state)

```
(:predicates
 (clear ?x)
 (handempty)
 (on ?x ?y)
 (ontable ?x)
 (holding ?x))

(:action pick-up
 :parameters (?x)
 :precondition (and
  (clear ?x)
  (handempty)
  (ontable ?x))
 :effect (and
  (holding ?x)
  (not (handempty))
  (not (clear ?x))
  (not (ontable ?x))))

(:action put-down
 :parameters (?x)
 :precondition (and
  (holding ?x))
 :effect (and
  (handempty)
  (clear ?x)
  (ontable ?x)
  (not (holding ?x))))

(:action stack
 :parameters (?x ?y)
 :precondition (and
  (clear ?y)
  (holding ?x))
 :effect (and
  (handempty)
  (on ?x ?y)
  (clear ?x)
  (not (clear ?y))
  (not (holding ?x))))

(:action unstack
 :parameters (?x ?y)
 :precondition (and
  (clear ?x)
  (handempty)
  (on ?x ?y))
 :effect (and
  (clear ?y)
  (holding ?x)
  (not (handempty))
  (not (on ?x ?y))
  (not (clear ?x))))
```

Listing 1: **STRIPS models extracted in `blocksworld`.**

(a) LS (applicability)

```
(:predicates
 (at ?x ?y)
 (at-ferry ?x)
 (empty)
 (is_car ?x)
 (is_port ?x)
 (p4 ?x))

(:action board
 :parameters (?x ?y)
 :precondition (and
  (at ?x ?y)
  (at-ferry ?y)
  (empty)
  (is_car ?x)
  (is_port ?y))
 :effect (and
  (at ?y ?x)
  (at-ferry ?x)
  (not (at ?x ?y))
  (not (empty))
  (not (is_car ?y))))

(:action debark
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?y)
  (is_car ?x)
  (is_port ?y)
  (at-ferry ?x))
 :effect (and
  (at ?x ?y)
  (empty)
  (at ?y ?x)
  (is_car ?y)
  (not (at-ferry ?x)))
   )

(:action sail
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?x)
  (is_port ?x)
  (is_port ?y))
 :effect (and
  (at-ferry ?y)
  (is_car ?y)
  (not (at-ferry ?x))
  (not (is_car ?x))))
```

(b) LS (final-state)

```
(:predicates
 (at ?x ?y)
 (at-ferry ?x)
 (empty)
 (is_car ?x)
 (is_port ?x)
 (in ?x))

(:action board
 :parameters (?x ?y)
 :precondition (and
  (at ?x ?y)
  (at-ferry ?y)
  (empty)
  (is_car ?x)
  (is_port ?y))
 :effect (and
  (in ?x)
  (not (at ?x ?y))
  (not (empty))))

(:action debark
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?y)
  (is_car ?x)
  (is_port ?y)
  (in ?x))
 :effect (and
  (at ?x ?y)
  (empty)
  (not (in ?x))))

(:action sail
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?x)
  (is_port ?x)
  (is_port ?y))
 :effect (and
  (at-ferry ?y)
  (not (at-ferry ?x)))
   )
```

(c) LB (final-state)

```
(:predicates
 (at ?x ?y)
 (at-ferry ?x)
 (empty)
 (is_car ?x)
 (is_port ?x)
 (in ?x))

(:action board
 :parameters (?x ?y)
 :precondition (and
  (at ?x ?y)
  (at-ferry ?y)
  (empty)
  (is_car ?x)
  (is_port ?y))
 :effect (and
  (in ?x)
  (not (at ?x ?y))
  (not (empty))))

(:action debark
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?y)
  (is_car ?x)
  (is_port ?y)
  (in ?x))
 :effect (and
  (at ?x ?y)
  (empty)
  (not (in ?x))))

(:action sail
 :parameters (?x ?y)
 :precondition (and
  (at-ferry ?x)
  (is_port ?x)
  (is_port ?y))
 :effect (and
  (at-ferry ?y)
  (not (at-ferry ?x)))
   )
```

Listing 2: **STRIPS models extracted in `ferry`.**

(a) LS (applicability)

```
(:predicates
 (at ?x ?y)
 (in-city ?x ?y)
 (is_airplane ?x)
 (is_airport ?x)
 (is_city ?x)
 (is_location ?x)
 (is_package ?x)
 (is_truck ?x)
 (is_vehicle ?x)
 (p9 ?x ?y))

(:action drive
 :parameters (?x ?y ?z
     ?u)
 :precondition (and
  (in-city ?y ?u)
  (in-city ?z ?u)
  (is_city ?u)
  (is_truck ?x)
  (is_vehicle ?x)
  (at ?x ?y))
 :effect (and
  (at ?x ?z)
  (at ?x ?u)
  (at ?y ?x)
  (at ?z ?u)
  (in-city ?x ?z)
  (in-city ?y ?u)
  (is_airplane ?y)
  (is_city ?u)
  (is_location ?z)
  (is_truck ?x)
  (not (at ?x ?y))
  (not (in-city ?x ?y)
   )))

(:action fly
 :parameters (?x ?y ?z
    )
 :precondition (and
  (is_airplane ?x)
  (is_airport ?z)
  (is_location ?y)
  (is_vehicle ?x)
  (at ?x ?y))
 :effect (and
  (at ?x ?z)
  (at ?y ?x)
  (at ?y ?z)
  (at ?z ?x)
  (at ?z ?y)
  (in-city ?x ?z)
  (is_airplane ?y)
  (is_airplane ?z)
  (is_airport ?z)
  (is_location ?y)
  (is_location ?z)
  (not (at ?x ?y))))
```

(b) LS (final-state)

```
(:predicates
 (at ?x ?y)
 (in-city ?x ?y)
 (is_airplane ?x)
 (is_airport ?x)
 (is_city ?x)
 (is_location ?x)
 (is_package ?x)
 (is_truck ?x)
 (is_vehicle ?x)
 (in ?x ?y))

(:action drive
 :parameters (?x ?y ?z
     ?u)
 :precondition (and
  (in-city ?y ?u)
  (in-city ?z ?u)
  (is_city ?u)
  (is_truck ?x)
  (is_vehicle ?x)
  (at ?x ?y)
  (is_location ?y)
  (is_location ?z))
 :effect (and
  (at ?x ?z)
  (not (at ?x ?y))))

(:action fly
 :parameters (?x ?y ?z
    )
 :precondition (and
  (is_airplane ?x)
  (is_airport ?z)
  (is_location ?y)
  (is_vehicle ?x)
  (at ?x ?y)
  (is_airport ?y)
  (is_location ?z))
 :effect (and
  (at ?x ?z)
  (not (at ?x ?y))))
```

(c) LB (final-state)

```
(:predicates
 (at ?x ?y)
 (in-city ?x ?y)
 (is_airplane ?x)
 (is_airport ?x)
 (is_city ?x)
 (is_location ?x)
 (is_package ?x)
 (is_truck ?x)
 (is_vehicle ?x)
 (in ?x ?y))

(:action drive
 :parameters (?x ?y ?z
     ?u)
 :precondition (and
  (in-city ?y ?u)
  (in-city ?z ?u)
  (is_city ?u)
  (is_truck ?x)
  (is_vehicle ?x)
  (is_location ?y)
  (is_location ?z))
 :effect (and
  (at ?x ?z)
  (not (at ?x ?y))))

(:action fly
 :parameters (?x ?y ?z
    )
 :precondition (and
  (is_airplane ?x)
  (is_airport ?z)
  (is_location ?y)
  (is_vehicle ?x)
  (is_airport ?y)
  (is_location ?z))
 :effect (and
  (at ?x ?z)
  (not (at ?x ?y))))
```

Listing 3: **STRIPS models extracted in `logistics`.**

(a) LS (applicability)

```
(:action load
 :parameters (?x ?y ?z
   )
 :precondition (and
  (is_location ?z)
  (is_package ?y)
  (is_vehicle ?x)
  (at ?x ?z)
  (at ?y ?z))
 :effect (and
  (at ?x ?z)
  (at ?y ?x)
  (at ?z ?x)
  (at ?z ?y)
  (in-city ?x ?y)
  (in-city ?x ?z)
  (in-city ?y ?x)
  (is_airplane ?z)
  (is_city ?x)
  (is_vehicle ?x)
  (not (at ?y ?z))
  (not (is_location ?y
    ))))

(:action unload
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?z)
  (is_location ?z)
  (is_package ?y)
  (in-city ?x ?z)
  (in-city ?y ?x)
  (is_city ?x))
 :effect (and
  (at ?y ?z)
  (at ?x ?z)
  (at ?y ?x)
  (at ?z ?x)
  (at ?z ?y)
  (is_airplane ?z)
  (is_location ?z)
  (not (in-city ?y ?x)
    )
  (not (is_location ?x
    ))))
```

(b) LS (final-state)

```
(:action load
 :parameters (?x ?y ?z
   )
 :precondition (and
  (is_location ?z)
  (is_package ?y)
  (is_vehicle ?x)
  (at ?x ?z)
  (at ?y ?z))
 :effect (and
  (in ?y ?x)
  (not (at ?y ?z))))

(:action unload
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?z)
  (is_location ?z)
  (is_package ?y)
  (in ?y ?x)
  (is_vehicle ?x))
 :effect (and
  (at ?y ?z)
  (not (in ?y ?x))))
```

(c) LB (final-state)

```
(:action load
 :parameters (?x ?y ?z
   )
 :precondition (and
  (is_location ?z)
  (is_package ?y)
  (is_vehicle ?x))
 :effect (and
  (in ?y ?x)
  (not (at ?y ?z))))

(:action unload
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?z)
  (is_location ?z)
  (is_package ?y)
  (in ?y ?x)
  (is_vehicle ?x))
 :effect (and
  (at ?y ?z)
  (not (in ?y ?x))))
```

Listing 4: **Continuation of Listing 3.**

(a) LS (applicability)

```
(:predicates
 (at ?x ?y)
 (empty ?x)
 (is_position ?x)
 (is_tile ?x)
 (neighbor ?x ?y))

(:action move
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?y)
  (empty ?z)
  (is_position ?y)
  (is_tile ?x)
  (neighbor ?y ?z)
  (neighbor ?z ?y))
 :effect (and
  (at ?x ?z)
  (empty ?y)
  (at ?y ?z)
  (at ?z ?y)
  (neighbor ?x ?y)
  (neighbor ?x ?z)
  (neighbor ?y ?z)
  (not (at ?x ?y))
  (not (empty ?z))
  (not (is_position ?x
   ))))
```

(b) LS (final-state)

```
(:predicates
 (at ?x ?y)
 (empty ?x)
 (is_position ?x)
 (is_tile ?x)
 (neighbor ?x ?y))

(:action move
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?y)
  (empty ?z)
  (is_position ?y)
  (is_tile ?x)
  (neighbor ?y ?z)
  (neighbor ?z ?y)
  (is_position ?z))
 :effect (and
  (at ?x ?z)
  (empty ?y)
  (not (at ?x ?y))
  (not (empty ?z))))
```

(c) LB (final-state)

```
(:predicates
 (at ?x ?y)
 (empty ?x)
 (is_position ?x)
 (is_tile ?x)
 (neighbor ?x ?y))

(:action move
 :parameters (?x ?y ?z
   )
 :precondition (and
  (at ?x ?y)
  (empty ?z)
  (is_position ?y)
  (is_tile ?x)
  (neighbor ?y ?z)
  (neighbor ?z ?y)
  (is_position ?z))
 :effect (and
  (at ?x ?z)
  (empty ?y)
  (not (at ?x ?y))
  (not (empty ?z))))
```

Listing 5: **STRIPS models extracted in `npuzzle`.**

(a) LS (applicability)

```
(:predicates
 (at ?x)
 (connected ?x ?y)
 (free ?x))

(:action move
 :parameters (?x ?y)
 :precondition (and
  (at ?x)
  (connected ?x ?y)
  (connected ?y ?x)
  (free ?x)
  (free ?y))
 :effect (and
  (at ?y)
  (not (at ?x))))
```

(b) LS (final-state)

```
(:predicates
 (at ?x)
 (connected ?x ?y)
 (free ?x))

(:action move
 :parameters (?x ?y)
 :precondition (and
  (at ?x)
  (connected ?x ?y)
  (connected ?y ?x)
  (free ?x)
  (free ?y))
 :effect (and
  (at ?y)
  (not (at ?x))))
```

(c) LB (final-state)

```
(:predicates
 (at ?x)
 (connected ?x ?y)
 (free ?x))

(:action move
 :parameters (?x ?y)
 :precondition (and
  (at ?x)
  (connected ?x ?y)
  (connected ?y ?x)
  (free ?x)
  (free ?y))
 :effect (and
  (at ?y)
  (not (at ?x))))
```

Listing 6: **STRIPS models extracted in `maze`.**