From Next Token Prediction to (STRIPS) World Models – Preliminary Results

Carlos Núñez-Molina[1], **Vicenç Gómez**[2], **Hector Geffner**[1]

[1]RWTH Aachen University, Germany
[2]Universitat Pompeu Fabra, Spain
carlos.nunez@ml.rwth-aachen.de, vicen.gomez@upf.edu, hector.geffner@ml.rwth-aachen.de

Abstract

We consider the problem of learning propositional STRIPS world models from action traces alone, using a deep learning architecture (transformers) and gradient descent. The task is cast as a supervised next token prediction problem where the tokens are the actions, and an action a may follow an action sequence if the hidden effects of the previous actions do not make an action precondition of a false. We show that a suitable transformer architecture can faithfully represent propositional STRIPS world models, and that the models can be learned from sets of random valid (positive) and invalid (negative) action sequences alone. A number of experiments are reported.

Introduction

A number of recent works have considered whether LLMs, and in particular, transformer architectures, can learn predictive world models as a result of learning to predict the next token "autoregressively"[1]. The question is important because true understanding involves the ability to make meaningful predictions from a model, and it is far from clear whether LLMs exhibit true understanding or learn world models (Yildirim and Paul 2024; Vafa et al. 2024a).

In order to test the hypothesis that transformer networks can learn world models when predicting the next token, Kenneth Li and co-authors trained a transformer network called "OthelloGPT," to input sequences of legal moves, and output a next legal move (Li et al. 2023). Othello is played on a grid composed of a number of cells, and in order to check whether the transformer network learned a "world model", the authors check whether the final embeddings of the board cells that have been learned encode the state of the cells during the game: namely, whether the cell contains a black piece, a white piece, or is empty. This was done by testing whether it was possible to predict these cell attributes from each cell embedding alone.[2]

The problem of learning world models from next token prediction requires learning a state representation and the ways in which the states change. Transformers have been shown to be capable of learning such state representations, yet these representations are often incoherent and do not accurately reflect the underlying world model, thus breaking when applied to downstream tasks (Vafa et al. 2024b).

This paper is aimed at relating the tasks of next token prediction and world-model learning using transformers in the very concrete setting of STRIPS world models. The paper has two goals. On the one hand, it aims to contribute to this research thread by considering world models of a specific type, namely propositional STRIPS world models with tokens representing propositional STRIPS actions. The question then is whether next token prediction in this setting can yield STRIPS world model representations. The second goal is to contribute to the algorithms for learning domain-independent planning models from data, in particular, when the data contains information about actions and no information about states. In the setting of *lifted* STRIPS world models, this problem has been recently solved by the algorithm SIFT (Gösgens, Jansen, and Geffner 2024), yet the potential advantage of a transformer-based approach to model learning is the ability to handle other types of data (e.g., images instead of STRIPS actions and states), along with other extensions such as numerical information, noisy and incomplete traces, etc.

The paper is structured as follows. We first present related work and the relevant background. Then, we formulate the main task, namely learning a Boolean *trace classifier* $f_M(\tau)$ using a set T of valid and invalid action traces τ drawn from a hidden world model M. Next, we show how the Boolean function $f_M(\tau)$ can be expressed in the formal B-RASP language, which provides a logical abstraction of transformers (Yang, Chiang, and Angluin 2024), and how the function can be learned from T using a particular transformer architecture, which we call the STRIPS Transformer. We use the resulting method in a number of experiments for learning various propositional STRIPS world models, discuss the results, and present the conclusions and future work.

[1]This description is a summary of the very interesting discussion in Melanie Mitchell's blog on "LLMs and World Models"; https://aiguide.substack.com/p/llms-and-world-models-part-2.

[2]Interestingly, a linear classifier did not work for this task initially, but a non-linear classifier did. Later on, a linear classifier was shown to perform this classification correctly too provided that the cell attribute labels "white" and "black" were changed to "the color of the pieces of the player that played last" and "the color of the pieces of the other player"; see https://www.neelnanda.io/mechanistic-interpretability/othello.

Related Work

This work relates to three research threads: understanding transformers in terms of the formal languages they can recognize, learning STRIPS planning models, and learning world-models from next token prediction. We discussed the last thread above, and focus below on the first two.

Transformers and RASP languages. Transformers are neural network architectures suited for processing sequences of tokens, that have revolutionized the area of natural language in AI first (Vaswani et al. 2017), and then computer vision (Dosovitskiy et al. 2021). A transformer layer maps a sequence of tokens, represented by real vectors, into a new sequence by means of two main opperations: *self-attention* that updates tokens by considering inputs from "similar" tokens, and multi-layered perceptrons (MLPs) that update each token individually. A simple programming language, called RASP, was introduced to capture the computations performed by a transformer at a more abstract level (Weiss, Goldberg, and Yahav 2021), and a boolean version of RASP, called B-RASP, has been proposed recently to characterize these computations in terms of a class of formal languages, specifically, the star-free languages (Yang, Chiang, and Angluin 2024). Interestingly, it has been shown that the language of valid action traces induced by a STRIPS planning domain is itself star-free (Lin and Bercher 2022), which suggests that B-RASP programs, and therefore hard-attention transformers, are expressive enough to recognize such domains. We build on this result in this work.

STRIPS Model learning. Algorithms for learning STRIPS models from traces featuring both actions and states have been considered in a number of works (Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019; Lamanna et al. 2025). Other algorithms like LOCM and SIFT manage to learn the models from action traces alone (Cresswell, McCluskey, and West 2013; Gösgens, Jansen, and Geffner 2024), and others from action names and the structure of the state graph (Bonet and Geffner 2020; Rodriguez et al. 2021). The proposed STRIPS transformer architecture is particularly related to a version of SIFT where actions have no parameters, and the explicit set of features is replaced by an smaller subset of parametric features. In most of these cases, however, the learned STRIPS models are lifted, and both LOCM and SIFT learned form positive traces only. On the other hand, few works have been aimed at learning STRIPS models in the deep learning setting (Asai et al. 2022; Xi, Gould, and Thiébaux 2024), and none of them do so using action traces alone, as in this work.

Background

STRIPS

Propositional STRIPS problems $P = \langle F, A, I, G\rangle$ are made of four parts: a set of atoms or propositions F, a subset of propositions $I \subseteq F$ defining those which are true in the initial situation, a subset $G \subseteq F$ of goals to be achieved, and a set A of actions a, each one defined by three subsets of propositions from F: the precondition list $pre(a)$, the add list of $add(a)$, and the delete list $del(a)$,

A problem $P = \langle F, A, I, G\rangle$ defines a state model $S(P) = \langle S, s_0, S_G, A, A(\cdot), f\rangle$ where the states $s \in S$ are subsets of F, the initial state s_0 is I, the goal states $s \in S_G$ include all the goal atoms, $A(s)$ is the set of applicable actions in s given by those in A whose preconditions $pre(a)$ are all (true) in s, and a deterministic state-transition function $f(a, s)$ that maps s into the state $s' = (s \cup add(a)) \setminus del(a)$ when $a \in A(s)$.

An action sequence $\tau : a_0, \ldots, a_n$ in P is a sequence of actions from A. The action sequence is applicable in a state s in $S(P)$ if there is a sequence of states $s_0, \ldots, s_{n+1}$ in $S(P)$ such that each action a_i is applicable in s_i, and s_{i+1} is the state that results from action a_i in s_i, i.e., $s_{i+1} = f(a_i, s_i)$, $i = 0, \ldots, n$. The action sequence τ is a plan for P if it is applicable in the initial state of P and the resulting state s_{n+1} is a goal state.

A propositional STRIPS *domain* or *action model* $M = \langle F, A\rangle$ will refer to a *set* of propositional STRIPS problems $P = \langle F, A, I, G\rangle$ sharing the same propositions F and actions A, possibly differing in the initial situation I or goal G. A hidden model M will be learned from traces τ obtained from instances P of M.

B-RASP

B-RASP is a boolean version of the RASP language (Weiss, Goldberg, and Yahav 2021) which aims at providing an abstract representation of masked, hard-attention transformers, in order to facilitate their analysis (Yang, Chiang, and Angluin 2024). A masked, hard-attention transformer with a particular set of parameter values can be compiled into an equivalent B-RASP program and vice versa.

A B-RASP program uses boolean vectors of length n as its only datatype. It receives as input a string $w = (w_0, \ldots, w_n)$ composed of symbols w_i from a finite alphabet Σ, which is encoded as a set of initial vectors $\mathcal{I}_\sigma$, one for each $\sigma \in \Sigma$, where $\mathcal{I}_\sigma(i) = 1$ iff $w_i = \sigma$.

If these $|\Sigma|$ initial vectors $\mathcal{I}_\sigma$ are denoted as $P_1, \ldots, P_{|\Sigma|}$, a B-RASP program can be understood as defining new vectors P_t from previous vectors $P_1, \ldots, P_{t-1}$, through one of two operations:

Position-wise operations: Each element $P_t(i)$ of P_t is defined as a Boolean function over elements from $\{P_1(i), \ldots, P_{t-1}(i)\}$, $i = 1, \ldots, n$.

Attention operations: Each $P_t(i)$ is defined as

$$P_t(i) := \blacktriangle_j[M(i,j), S(i,j)]\ V(i,j)\ :\ D(i) \qquad (1)$$

where

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

For each $i \in [n]$, $P_t(i) := V(i, j_i)$, if j_i is the minimum (if $\blacktriangleleft_j$) or maximum (if $\blacktriangleright_j$) index $j \in [n]$ such that $M(i,j) = 1$ and $S(i,j) = 1$, provided that such element j_i exists. Otherwise, $P_t(i) := D(i)$. The intuition is that $P_t(i)$ is set to $V(i, j_i)$, a Boolean formula $\{P_1(i), \ldots, P_{t-1}(i)\} \cup \{P_1(j_i), \ldots, P_{t-1}(j_i)\}$, if $P_t(i)$ "attends" to j_i, and $P_t(i)$ is set to $D(i)$ otherwise, which is a Boolean formula over the elements $\{P_1(i), \ldots, P_{t-1}(i)\}$ only.

Multi-head attention can be simulated by summing the outputs of multiple attention operations like (1). This is particularly important for the compiled transformer, which can exploit it to parallelize computations across attention heads at the same depth.

A B-RASP program can be used to determine whether an input string w belongs to some language $L \subseteq \Sigma^*$, or to map input strings w into output strings $w' \in \Gamma^+$ of the same length. In the first case, w is encoded as the set of input vectors above, and it is regarded as accepted (in L) if $Z(n) = 1$ where Z is the last vector produced by the program. In the second case, w is encoded as the set of input vectors above, and w' is read from the last $|\Gamma|$ vectors Z_γ produced by the program, where $w'(i) = \gamma$ iff $Z_\gamma(i) = 1$, for $i = 1, \ldots, n$ and $\gamma \in \Gamma$.

Learning Task

The learning task is to uncover the propositions F and the structure of the actions A (preconditions and effects) from a set T of *positive* and *negative* action traces over A drawn from instances P of a hidden model $M = \langle F, A \rangle$. A trace over A refers to an action sequence $\tau = a_0, \ldots, a_n$ made up of actions from A and, since *no information about the initial states is given*, traces are deemed as positive or negative according to their *internal consistency*, defined as follows:

Definition 1 (Positive and negative traces)**.** *Given a propositional* STRIPS *domain $M = \langle F, A \rangle$ composed of problems P with actions A and propositions F, an action trace $\tau = (a_0, \ldots, a_n)$ is* positive *in M, written as $f_M(\tau) = 0$, if, for every atom $p \in \mathrm{pre}(a_i)$, one of the following holds:*

1. *no action a_j preceding a_i in τ affects p; i.e., $p \notin \{\mathrm{add}(a_j) \cup \mathrm{del}(a_j)\}$, or*
2. *the last action a_j preceding a_i in τ that affects p makes p true; i.e., $p \in \mathrm{add}(a_j)$.*

The trace τ is negative in M, written as $f_M(\tau) = 1$, if it is not positive.[3]

From the perspective of *next token prediction*, if we assume tokens to be the actions in M, a trace $\tau = (a_0, \ldots, a_n)$ is positive iff each action a_i is a *possible next action* given the previous actions $(a_0, \ldots, a_{i-1})$ in the trace, and negative otherwise.

Definition 2 (Learning task)**.** *Given a set of positive and negative traces T drawn from a hidden domain $M = \langle F, A \rangle$, the task is to learn a Boolean function f over all action sequences τ drawn from A (M-traces) that is equal to f_M; i.e., for which $f(\tau) = f_M(\tau)$ for any such τ.*

[3]We use the value 0 for positive traces and 1 for negative traces (instead of the other way around) for convenience.

Learning a function equal to $f_M(\tau)$ is equivalent to solving the next token prediction problem in the STRIPS setting. Indeed, $b \in A$ is a *possible next action* after a positive trace τ in M iff $f_M(\tau, b) = 0$, and it is not a possible next action after τ iff $f_M(\tau, b) = 1$.

We show next how the function $f_M(\tau)$ can be encoded as a B-RASP program for any STRIPS domain M, thus proving that a (masked) hard-attention transformer can encode $f_M(\tau)$. Then, we explain how to learn such a function from M-traces, using a specialized transformer architecture, which we call the STRIPS Transformer.

Encoding f_M in B-RASP

We explain next how to encode $f_M(\tau)$ into a B-RASP program $f_M^{\text{B-RASP}}$ for any STRIPS domain M. Let $f_M^{\text{B-RASP}}(\tau)$ denote the output of the program for a particular input τ.

Given some M-trace τ, this program returns $f_M^{\text{B-RASP}}(\tau) = 0$ if τ is positive, and $f_M^{\text{B-RASP}}(\tau) = 1$ if τ is negative. The input trace $\tau = (a_1, \ldots, a_n)$ is encoded using $|A|$ initial vectors I_a of size n, where $I_a(i) = 1$ iff $a_i = a$.

For each atom $p \in F$ in M, we define three vectors Q_p, K_p, V_p that depend on the particular M, and vectors Y_p and Y shared among domains:

- $Q_p(i)$ indicates whether $p \in \mathrm{pre}(a_i)$:

$$Q_p(i) := \bigvee_{a | p \in \mathrm{pre}(a)} I_a(i). \tag{2}$$

- $K_p(i)$ indicates whether p appears in either $\mathrm{add}(a_i)$ or $\mathrm{del}(a_i)$:

$$K_p(i) := \bigvee_{a | p \in \mathrm{add}(a) \cup \mathrm{del}(a)} I_a(i). \tag{3}$$

- $V_p(i)$ indicates whether $p \in \mathrm{del}(a_i)$:

$$V_p(i) := \bigvee_{a | p \in \mathrm{del}(a)} I_a(i). \tag{4}$$

- $Y_p(i)$ indicates whether $p \in \mathrm{pre}(a_i)$ *and* the most recent action before position i that affects p, deletes p. This is computed with an attention operation:

$$Y_p(i) := \blacktriangleright_j \ [j < i,\ Q_p(i) \wedge K_p(j)]\ V_p(j)\ :\ 0, \tag{5}$$

where we use $Q_p(i) \wedge K_p(j)$ as the score function, $V_p(j)$ as the value function, and 0 as the default value function. Intuitively, $Y_p(i) = 1$ means that action a_i contains a precondition p that is made false by the last preceding action in the trace that affects p (remember that $\blacktriangleright$ denotes max). Conversely, a value $Y_p(i) = 0$ means that either 1) a_i does not have p as precondition, 2) no preceding action in the trace affects p, or 3) the last preceding action that affects p, makes p true.

For each action $a_i \in \tau$, a_i will be applicable iff all of its preconditions $p \in \mathrm{pre}(a_i)$ are true. In that case, for each $p \in F$ we will have $Y_p(i) = 0$. Therefore, we combine all

Domain `simple`

Atoms:

p, q, r

Actions:

a:
pre(a) = [p, r]
add(a) = [q]
del(a) = [p, r]

b:
pre(b) = [q, r]
add(b) = [p]
del(b) = [q, r]

c:
pre(c) = []
add(c) = [r]
del(c) = []

(a)

	a	c	c	b	c	a
$I_{\mathtt{a}}$	1	0	0	0	0	1
$I_{\mathtt{b}}$	0	0	0	1	0	0
$I_{\mathtt{c}}$	0	1	1	0	1	0
$Q_{\mathtt{p}}$	1	0	0	0	0	1
$K_{\mathtt{p}}$	1	0	0	1	0	1
$V_{\mathtt{p}}$	1	0	0	0	0	1
$Y_{\mathtt{p}}$	0	0	0	0	0	0
$Q_{\mathtt{q}}$	0	0	0	1	0	0
$K_{\mathtt{q}}$	1	0	0	1	0	1
$V_{\mathtt{q}}$	0	0	0	1	0	0
$Y_{\mathtt{q}}$	0	0	0	0	0	0
$Q_{\mathtt{r}}$	1	0	0	1	0	1
$K_{\mathtt{r}}$	1	1	1	1	1	1
$V_{\mathtt{r}}$	1	0	0	1	0	1
$Y_{\mathtt{r}}$	0	0	0	0	0	0
Y	0	0	0	0	0	0
Z	0	0	0	0	0	**0**

(b)

	a	c	a	c	b	b
$I_{\mathtt{a}}$	1	0	1	0	0	0
$I_{\mathtt{b}}$	0	0	0	0	1	1
$I_{\mathtt{c}}$	0	1	0	1	0	0
$Q_{\mathtt{p}}$	1	0	1	0	0	0
$K_{\mathtt{p}}$	1	0	1	0	1	1
$V_{\mathtt{p}}$	1	0	1	0	0	0
$Y_{\mathtt{p}}$	0	0	1	0	0	0
$Q_{\mathtt{q}}$	0	0	0	0	1	1
$K_{\mathtt{q}}$	1	0	1	0	1	1
$V_{\mathtt{q}}$	0	0	0	0	1	1
$Y_{\mathtt{q}}$	0	0	0	0	0	1
$Q_{\mathtt{r}}$	1	0	1	0	1	1
$K_{\mathtt{r}}$	1	1	1	1	1	1
$V_{\mathtt{r}}$	1	0	1	0	1	1
$Y_{\mathtt{r}}$	0	0	0	0	0	1
Y	0	0	1	0	0	1
Z	1	1	1	1	1	**1**

(c)

Figure 1: Vectors produced by the B-RASP program $f_M^{\text{B-RASP}}$ for the two traces τ^+ and τ^- shown on top, drawn from the model M shown on the left, named `simple`. The value of $f_M^{\text{B-RASP}}(\tau)$ is given by the last entry of the last vector, i.e., $Z(n=6)$ (marked in bold), and for both traces, $f_M^{\text{B-RASP}}(\tau) = f_M(\tau)$. (a) The hidden STRIPS domain `simple`. (b) Vectors produced for computing $f_M^{\text{B-RASP}}(\tau^+)$. (c) Vectors produced for computing $f_M^{\text{B-RASP}}(\tau^-)$. The trace τ^- contains two inapplicable actions (marked in red): the second occurrence of `a`, for which the precondition p is false (see how $Y_p(3) = 1$), and the last occurrence of `b`, for which the preconditions q and r are false (see how $Y_q(6) = Y_r(6) = 1$). Finally, we observe that $Y(3) = 1$ and $Y(6) = 1$, meaning that $a_3 = a$ and $a_6 = b$ are inapplicable actions, so τ^- is a negative trace ($Y(6) = 1$).

the vectors $Y_p(i)$ into a single vector $Y(i)$ so that $Y(i) = 0$ iff a_i is applicable:

$$Y(i) := \bigvee_{p \in F} Y_p(i). \tag{6}$$

We note that the vectors $Y_p(i)$ for each p are computed with separate attention operations ("heads") and these computations can be done in parallel.

Finally, the trace τ will be positive (i.e., valid) iff every action $a_i \in \tau$ is applicable, i.e., iff $Y(i) = 0$ for every $i \in [n]$. This is encoded into the output vector Z, which is computed as follows:

$$Z(i) :=\blacktriangleright_j \ [1,\ Y(j)]\ 1\ :\ 0. \tag{7}$$

This vector Z will contain a value of 0 in every position i if the trace τ is positive, and a value of 1 if τ is negative. The output of the B-RASP program $f_M^{\text{B-RASP}}$ is given by the last position of this vector, i.e., $f_M^{\text{B-RASP}}(\tau) = Z(n)$. Therefore, this program computes the function $f_M(\tau)$:

Theorem 3. *Let τ be an action sequence drawn from M. Then, $f_M^{\text{B-RASP}}(\tau) = f_M(\tau)$.*

Example. Fig. 1(a) shows a simple STRIPS model M with three actions, $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, and three atoms, $F = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$. Positive traces alternate between `a` and `b`, with one or more occurrences of `c` in-between. Fig. 1(b) shows the execution of the B-RASP program $f_M^{\text{B-RASP}}$ on a positive trace τ^+ (shown on top), while Fig. 1(c) shows the execution for a negative trace τ^- (shown on top as well). It can be observed that $f_M^{\text{B-RASP}}(\tau^+) = 0$ and $f_M^{\text{B-RASP}}(\tau^-) = 1$, in agreement with f_M. The computations are depicted in five blocks separated by horizontal lines: the first block encodes the inputs, the inner three blocks encode computations that can be perfomed in parallel from the input block, as they correspond to three "attention heads", and the last block encodes the outputs from them.

The STRIPS Transformer

B-RASP programs can be compiled into and learned via transformers (Yang, Chiang, and Angluin 2024). Rather than using standard hard-attention transformers, we will define a specialized transformer architecture (also based on hard-attention) for computing a parametric function $f_\theta(\tau)$, where θ denotes a particular set of transformer parameters. The parameters θ will be learned from a set of positive and negative traces T drawn from the corresponding domain M. Our architecture guarantees that there exists a set of parameter values θ^* such that function $f_{\theta^*}(\tau)$ is equivalent to the ground-truth function $f_M(\tau)$ over all traces. To learn θ^*, the set of training traces T must be informative enough.

We first describe the proposed architecture assuming access to the hidden model M, and hence to the optimal parameters θ^*, and later relax this assumption. The proposed

STRIPS Transformer can be seen as a differentiable version of the previous B-RASP program, where operations over Boolean vectors are replaced by (equivalent) operations over real-valued tensors.

Let $l = 1, \ldots, |F|$ index atoms and $m = 1, \ldots, |A|$ index actions. Define the parameters as a real-valued tensor $\theta \in [0,1]^{|F| \times |A| \times 3}$ as follows:

$$\theta(l, m, 1) := \begin{cases} 1 & \text{if } p_l \in \mathrm{pre}(a_m) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$\theta(l, m, 2) := \begin{cases} 1 & \text{if } p_l \in \mathrm{add}(a_m) \cup \mathrm{del}(a_m) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\theta(l, m, 3) := \begin{cases} 1 & \text{if } p_l \in \mathrm{del}(a_m) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Each parameter represents an analogue of the logical conditions defined for queries (8), keys (9) and values (10) in the B-RASP formulation introduced earlier.

Note that these parameter values correspond to the optimal configuration θ^*, but can only be defined when the model M is known. In the next section, we will describe how to learn θ from the traces when M is unknown.

Architecture

The STRIPS Transformer is a multi-head architecture that applies one attention head l per domain atom p_l. Each head l operates as follows:

- **QKV projection.** For each action a_i, let m be its index in θ. The vectors $Q_{p_l}, K_{p_l}, V_{p_l} \in [0,1]^n$ are computed as:

$$Q_{p_l}(i) := \theta(l, m, 1), \tag{11}$$

$$K_{p_l}(i) := \theta(l, m, 2), \tag{12}$$

$$V_{p_l}(i) := \theta(l, m, 3), \tag{13}$$

  where the θ parameters in $[0,1]^{|F| \times |A| \times 3}$ are to be learned. These are the real-valued analogues of the B-RASP vectors in Eqs. 2–4.

- **Attention Scores.** The attention scores are computed as

$$S_{p_l} := Q_{p_l} \cdot K_{p_l}^{\top}, \tag{14}$$

  resulting in a matrix $S_{p_l} \in [0,1]^{n \times n}$, with each entry $S_{p_l}(i,j)$ quantifying how much action a_j affects action a_i regarding atom p_l. This corresponds to $Q_p(i) \wedge K_p(j)$ in Eq. 5.

- **Strict future masking.** We apply strict future masking by setting $S_{p_l}(i,j) := 0$ if $j \geq i$, which encodes the fact that actions a_i can only be affected by preceding actions a_j ($j < i$). This corresponds to the $j < i$ mask in Eq. 5.

- **Stick-breaking attention.** To implement hard-attention with scalar scores, we use stick-breaking attention (Tan et al. 2025). Given S_{p_l}, this computes:

$$S'_{p_l}(i,j) := S_{p_l}(i,j) \cdot \prod_{k=j+1}^{n} \big(1 - S_{p_l}(i,k)\big). \tag{15}$$

  This ensures that attention focuses on the rightmost high-scoring position preceding a_i, in agreement with the $\blacktriangleright_j$ operator in B-RASP used in Eq. 5. Note that no explicit $\max$ operation is required.

- **Value aggregation.** The output vector for head p_l, denoted $(y_1, \ldots, y_n)_{p_l} \in [0,1]^n$, is computed as the matrix-vector product

$$y_{p_l} := S'_{p_l} \cdot V_{p_l}. \tag{16}$$

  Each entry $y_{p_l}(i)$ indicates the degree to which action a_i is inconsistent with respect to p_l. This corresponds to Y_p in Eq. 5.

Finally, the outputs of all heads are combined into a single vector y using a real-valued analogue of the logical OR. This step, equivalent to Eq. 6, indicates for each position i the degree to which the action a_i in the trace is inconsistent:

$$y(i) := 1 - \prod_{l=1}^{|F|} \big(1 - y_{p_l}(i)\big). \tag{17}$$

To compute $f_\theta(\tau)$, an additional OR is performed over all the entries in y, which aggregates the inconsistency of the different actions $a_i \in \tau$. This corresponds to Z in Eq. 7:

$$f_\theta(\tau) := 1 - \prod_{i=1}^{n} \big(1 - y(i)\big). \tag{18}$$

Example. Figure 3 in the Appendix illustrates the above computations for the example in Figure 1, using the optimal parameterization θ^*.

The first key property of the proposed architecture is that the STRIPS transformer can capture any STRIPS model:

Theorem 4 (From STRIPS to Transformer)**.** *Let $M = \langle F, A \rangle$ be any propositional* STRIPS *model and let $\theta^* \in \{0,1\}^{|F| \times |A| \times 3}$ be as in Eqs. 8–10 above:*

- *$\theta^*(l, m, 1) = 1$ if $p_l \in \mathrm{pre}(a_m)$; 0 otherwise,*
- *$\theta^*(l, m, 2) = 1$ if $p_l \in \mathrm{add}(a_m) \cup \mathrm{del}(a_m)$; 0 otherwise,*
- *$\theta^*(l, m, 3) = 1$ if $p_l \in \mathrm{del}(a_m)$; 0 otherwise.*

Then, for any M-trace τ, $f_{\theta^*}(\tau) = f_M(\tau)$.

The second key property is that the learned *next token prediction* function f_θ yields an interpretable STRIPS model, following a suitable mapping of the vectors of learned real parameters θ into 0's and 1's, whose correctness can be analyzed. For this, let us define the binarized version of θ as:

$$\bar{\theta}(l, m, k) = [\![\theta(l, m, k) \geq 0.5]\!], \tag{19}$$

for all $l = 1, \ldots, |F|$, $m = 1, \ldots, |A|$, and $k = 1, 2, 3$. Note that when $\theta \in \{0,1\}^{|F| \times |A| \times 3}$, all computations in the STRIPS Transformer reduce to Boolean operations, so that $f_{\bar{\theta}}(\tau)$ is a Boolean classifier which outputs either 0 or 1.

Let us define the STRIPS model $M_{\bar{\theta}}$ determined from the binarized vector $\bar{\theta}$ of learned parameters of the STRIPS transformer as follows:

Definition 5. *The* STRIPS *model $M_{\bar{\theta}}$ defined from the hidden* STRIPS *model $M = \langle F, A \rangle$ and the learned parameters θ from the corresponding* STRIPS *transformer with $|F|$ heads l, has actions $a \in A$ with preconditions, add, and delete effects given as:*

- $\mathrm{pre}(a_m) = \{p_l \mid \bar{\theta}(l, m, 1)\}$,
- $\mathrm{add}(a_m) = \{p_l \mid \bar{\theta}(l, m, 2) \wedge \neg\bar{\theta}(l, m, 3)\}$,
- $\mathrm{del}(a_m) = \{p_l \mid \bar{\theta}(l, m, 2) \wedge \bar{\theta}(l, m, 3)\}$.

The second result can be expressed as follows:

Theorem 6 (From Transformer to STRIPS)**.** *Let $f_{\bar{\theta}}$ be the Boolean function obtained from the* STRIPS *transformer after learning, where $\bar{\theta}$ is defined from the learned vector of real parameters θ as in Eq. 19. Then, if $f_{\bar{\theta}}(\tau) = f_M(\tau)$ for all M-traces τ, the learned* STRIPS *model $M_{\bar{\theta}}$ defined by Def. 5 is equivalent to M (i.e., has the same positive and negative traces).*

Clearly, we cannot determine, in general, if $f_{\bar{\theta}}(\tau) = f_M(\tau)$ for all action sequences τ drawn from M, but we can evaluate this condition experimentally over random traces τ. The theorem implies that if this test is positive, the learned STRIPS model $M_{\bar{\theta}}$ that can be extracted from the learned STRIPS transformer weights is equivalent to the hidden model M.

Training the STRIPS Transformer

The operations described in Eqs. (11)–(18) are fully differentiable, meaning that the STRIPS Transformer can be trained using gradient descent.

Given a set of positive and negative training traces $T = T^+ \cup T^-$ extracted from a hidden domain M, our objective is to learn parameters θ such that $f_\theta(\tau) = f_M(\tau)$ for all traces $\tau \in T$.

We assume a supervised setting in which positive traces $\tau^+ \in T^+$ consist entirely of applicable actions, while negative traces $\tau^- \in T^-$ contain a single inapplicable action at the final position. Thus, for a negative trace $\tau^- = (a_1, \ldots, a_n)$, we assume that $a_1, \ldots, a_{n-1}$ are applicable and only a_n is inapplicable.

For clarity, we let $y_i = f_\theta(\tau_{1:i})$ denote the predicted probability that the i-th action a_i is inapplicable, conditioned on the prefix $\tau_{1:i-1}$. This corresponds to Eq. (17) and can be computed in parallel for the entire trace.

To account for the imbalance between applicable and inapplicable actions, we adopt the focal loss (Lin et al. 2017) instead of the standard binary cross-entropy loss. The focal loss includes two hyperparameters: $\alpha \in (0, 1)$, which balances the importance of the two classes, and $\gamma > 0$, which controls the strength of the focusing effect. It reduces to the standard binary cross-entropy loss for $\alpha = 0.5$ and $\gamma = 0$. We use $\alpha = 0.9$ and $\gamma = 3$.

The total loss is computed as:

$$\mathcal{L}(\theta; T) = \frac{1}{|T|}\left(\sum_{\tau^+ \in T^+} \mathcal{L}^+(\theta; \tau^+) + \sum_{\tau^- \in T^-} \mathcal{L}^-(\theta; \tau^-)\right), \tag{20}$$

where the losses for positive traces τ^+, which must result in outputs $y_i = 0$ for all i, are defined as:

$$\mathcal{L}^+(\theta; \tau^+) = -\frac{1}{|\tau^+|}\sum_{i=1}^{|\tau^+|}(1-\alpha)\,(y_i)^\gamma \log(1-y_i), \tag{21}$$

while the losses for negative traces τ^-, which must result in outputs $y_i = 0$ except for $y_{|\tau^-|} = 1$, are defined as:[4]

$$\mathcal{L}^-(\theta; \tau^-) = -\frac{1}{|\tau^-|}\sum_{i=1}^{|\tau^-|-1}(1-\alpha)\,(y_i)^\gamma \log(1-y_i) - \frac{1}{|\tau^-|}\,\alpha\,(1-y_{|\tau^-|})^\gamma \log(y_{|\tau^-|}). \tag{22}$$

Although the model is trained using real-valued parameters θ, we use the binarized parameters $\bar{\theta}$ as defined in Eq. (19) to classify unseen traces τ' as positive ($f_{\bar{\theta}}(\tau') = 0$) or negative ($f_{\bar{\theta}}(\tau') = 1$). Moreover, the corresponding STRIPS model $M_{\bar{\theta}}$ is obtained from these binarized parameters as in Def. 5.

Experiments

We now evaluate experimentally the ability of our approach to recover symbolic STRIPS models from action traces. Specifically, we are interested in addressing the following questions:

- Can the proposed method learn accurate symbolic models from positive and negative traces?
- How do dataset size and domain complexity affect learning and generalization?
- How interpretable are the learned models?

We consider five propositional planning domains. For each domain, we generate a dataset of traces to train the transformer and evaluate its performance on a separate set of test traces. We assess the quality of the learned domain by measuring classification accuracy and comparing the symbolic model extracted from the learned parameters to the hidden ground-truth domain. The domains are:

- The `simple` domain of Figure 1, containing $|F| = 3$ propositions and $|A| = 3$ actions.
- The grounded `blocksworld` domain, which involves reassembling a set of stackable blocks into a target configuration with a gripper. We use two grounded versions: `blocksworld-2b` ($|F| = 9$, $|A| = 8$) and `blocksworld-3b` ($|F| = 16$, $|A| = 18$), containing two and three blocks, respectively.
- The grounded `ferry` domain, where a ferry is used to transport cars between different ports. We use two grounded versions: `ferry-1c` ($|F| = 6$, $|A| = 6$) and `ferry-2c` ($|F| = 9$, $|A| = 10$), containing two ports and a single car, and two ports and two cars, respectively.

[4] Recall that, for training, a trace $a_1, \ldots, a_n$ is negative if the sub-trace $a_1, \ldots, a_{n-1}$ is positive, and action a_n is inapplicable (i.e., it is not a possible next action after a_{n-1}).

Size	blocksworld-2b		blocksworld-3b	
	train	test	train	test
50	$1.0 \pm .0$ (1.0)	$.933 \pm .026$ (.929)	$1.0 \pm .0$ (1.0)	$.609 \pm .062$ (.72)
100	$1.0 \pm .0$ (1.0)	$.962 \pm .014$ (.943)	$1.0 \pm .0$ (1.0)	$.845 \pm .030$ (.843)
200	$1.0 \pm .0$ (1.0)	$.996 \pm .004$ (.996)	$1.0 \pm .0$ (1.0)	$.931 \pm .015$ (.948)
500	$1.0 \pm .0$ (1.0)	$.998 \pm .001$ (.999)	$1.0 \pm .0$ (1.0)	$.982 \pm .007$ (.973)
1000	$1.0 \pm .0$ (1.0)	$.998 \pm .002$ (**1.0**)	$1.0 \pm .0$ (1.0)	$.993 \pm .002$ (.993)
2000	$1.0 \pm .0$ (1.0)	$.998 \pm .002$ (**1.0**)	$1.0 \pm .0$ (1.0)	$.998 \pm .001$ (**1.0**)

Size	ferry-1c		ferry-2c		simple	
	train	test	train	test	train	test
50	$.996 \pm .013$ (1.0)	$.971 \pm .010$ (.982)	$1.0 \pm .0$ (1.0)	$.873 \pm .032$ (.893)	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)
100	$.995 \pm .007$ (1.0)	$.955 \pm .015$ (.977)	$.997 \pm .009$ (1.0)	$.943 \pm .013$ (.929)	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)
200	$1.0 \pm .0$ (1.0)	$.985 \pm .012$ (.975)	$1.0 \pm .0$ (1.0)	$.983 \pm .006$ (.968)	$.980 \pm .063$ (1.0)	$.983 \pm .052$ (**1.0**)
500	$1.0 \pm .0$ (1.0)	$1.0 \pm .001$ (**1.0**)	$.998 \pm .007$ (1.0)	$.993 \pm .009$ (.996)	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)
1000	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)	$1.0 \pm .0$ (1.0)	$.998 \pm .001$ (.998)	–	–
2000	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)	$1.0 \pm .0$ (1.0)	$1.0 \pm .0$ (**1.0**)	–	–

Table 1: Experimental results. Columns correspond to propositional domains, whereas rows correspond to training dataset sizes. For each domain and size, we show the training and test accuracy. Cells contain the average and std values across seeds. Training columns contain in parentheses the best training accuracy across seeds. Test columns contain in parentheses the test accuracy obtained with the seed that resulted in the best training accuracy, with the best value(s) in each column highlighted in bold. Figure 2 shows the domains learned on `simple` with a dataset size of 200, marked in blue on the table. Finally, we could not obtain results on the `simple` domain for dataset sizes 1000 and 2000, since this domain does not contain enough unique traces of length $n = 10$.

Data generation. For each domain, we generate a training and a test dataset composed of action traces sampled randomly from the set of propositional actions in the domain. The traces are generated from four initial states per domain: two used for training and two for testing. Training datasets are created with $50, 100, 200, 500, 1000$, or 2000 unique traces, except for the `simple` domain, where only up to 500 traces can be generated due to a limited number of unique traces. Each training dataset contains 80% negative and 20% positive traces, to compensate for class imbalance. Trace lengths are sampled uniformly from a fixed range: up to 10 actions for `simple`, 20 for `blocksworld-2b` and `ferry-1c`, and 30 for `blocksworld-3b` and `ferry-2c`. Test datasets contain 5000 positive and 5000 negative traces, all with lengths up to 50, to assess generalization to longer traces.

Training and evaluation. For these experiments, we assume that the correct number of atoms is known (this is a hyperparameter whose analysis we leave for future work). As a consequence, the model uses as many attention heads as there are propositions in the domain. Parameters θ are initialized uniformly between 0 and 1 and trained using the Rectified Adam (RAdam) optimizer (Liu et al. 2019) with a learning rate of 0.02 on batches containing 8 traces.

For a fixed domain and training dataset, we optimize θ using 10 different random seeds. Thus, the only source of variation across runs is the random initialization of the model parameters. For each seed, we obtain the binarized parameters $\bar{\theta}$ after training. The model accuracy is then evaluated on both the training and test datasets. We consider that it correctly classifies a trace $\tau = (a_1, \ldots, a_n)$ iff it correctly classifies each action $a_i \in \tau$, i.e., $f_{\bar{\theta}}(\tau_{1:i}) = f_M(\tau_{1:i})$ for every $i \in [n]$, where M is the ground-truth model.

Rather than using a validation set, we train the model for a fixed number of steps. We set this number to 100000, which is sufficiently large to ensure convergence to a (local) minimum. All experiments were run on a machine equipped with an Nvidia A10 GPU and an Intel Xeon Platinum 8352M CPU. However, our model can also be trained on a consumer-grade GPU or CPU. Finally, our code has been made publicly available in a GitHub repository.[5]

Results

Table 1 shows the experimental results. We observe that generalization improves as the size of the training set increases. This trend can be seen in the test accuracy columns, particularly in the values in parentheses, which indicate the test accuracy obtained with the best-performing training seed.

When the STRIPS Transformer achieves 100% training accuracy, it has learned a domain that correctly classifies all the training traces. As the size of the training dataset increases, maintaining 100% training accuracy becomes more indicative that the learned domain matches the hidden one. In the limit, if the dataset is sufficiently large and the model still classifies all traces correctly, the learned domain must coincide with the ground-truth domain, resulting in 100% test accuracy. Thus, two conditions are required to recover the hidden domain: (1) a sufficiently large and diverse training

[5] https://github.com/TheAeryan/strips-transformer

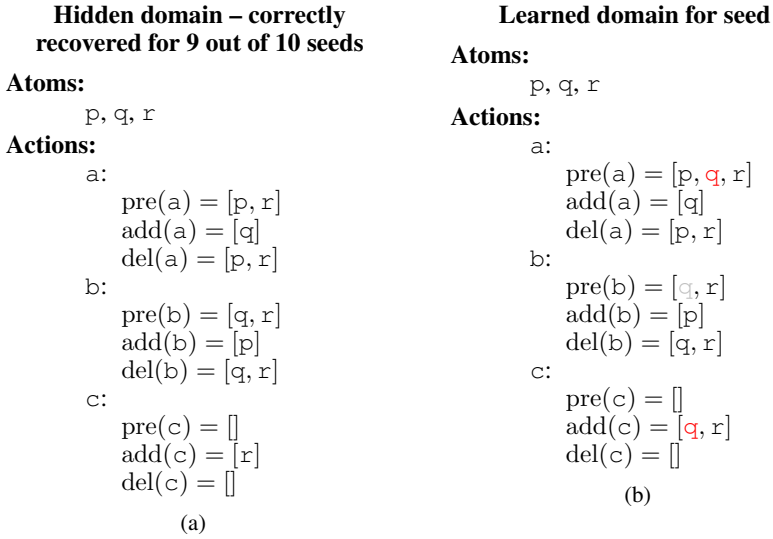

Figure 2: Examples of domains learned in `simple` for the training dataset with 200 samples. For comparison purposes, we renamed the propositions to `p`, `q`, `r` as in Figure 1. Incorrect preconditions, add effects and delete effects are highlighted in color: red for excess propositions and gray for those that are absent. (a) shows the domain learned by the STRIPS Transformer in 9 out of 10 seeds, which is equivalent to the hidden, ground-truth domain. As a result, it obtains both perfect training and test accuracy. (b) shows the domain learned for seed 5. This domain contains several incorrect preconditions and effects, thus resulting in 80% training accuracy and 83% test accuracy.

set, and (2) a transformer that achieves perfect classification on it. In this setting, the learned domain generalizes correctly to previously unseen traces of any length.

Also, the STRIPS Transformer successfully learns in all five domains. When the training dataset is large enough, it achieves perfect training accuracy for each random seed, which showcases its robustness. Additionally, it achieves 100% test accuracy for every domain, correctly classifying all the test traces despite them being considerably longer than the training ones. This indicates that the transformer has learned a domain equivalent to the hidden one used to generate the traces.

Figure 2 illustrates the learned domains for the `simple` domain using 200 training samples (corresponding to the blue entries in Table 1). In this case, the recovered domain matches the ground truth in nine out of ten seeds. When we increase the size of the training dataset to 500 samples, the transformer correctly learns the ground-truth domain for all ten seeds, as Table 1 shows.

Discussion

We have formulated the problem of learning propositional STRIPS world models from positive and negative action traces as a next token (action) prediction problem, and have addressed it using a specialized, STRIPS transformer architecture. For this, we have shown that the trace classification task can be expressed and captured in the B-RASP language, which can be compiled into hard-attention transformers. Our approach yields STRIPS models that are interpretable and which, in many cases, can be shown to be correct.

In the future, we want to be able to address the problem of learning *lifted* STRIPS models using transformers, following the success of the symbolic SIFT algorithm introduced recently (Gösgens, Jansen, and Geffner 2024). Indeed, there is a close relation between the STRIPS transformer and the SIFT algorithm when specialized in (propositional) domains where actions have no parameters. In this case, SIFT produces each of the possible domain features *exhaustively* (atoms with the actions that add and delete them), and checks the consistency of each one relative to the given traces, all of which are positive. In the STRIPS transformer, there is instead a smaller set of *parametric features*, each one associated with a transformer head, such that each parametric feature results in a consistent feature once the value of the parameters are learned and binarized.

SIFT is efficient and comes with a number of formal guarantees that the STRIPS transformer cannot match. Yet, the potential advantage of a transformer-based approach to model learning is the ability to handle other types of inputs in a robust manner. Our next task will be to bring these two threads closer together, by extending the STRIPS transformer so that it learns lifted STRIPS domains.

Acknowledgments

The research has been partially supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, by the European Research Council (ERC), Grant agreement No. No 885107, and by the Excellence Strategy of the Federal Government and the NRW state, Germany. This publication is also part of the action CNS2022-136178 financed by MCIN/AEI/10.13039/501100011033 and by the European Union Next Generation EU/PRTR.

Appendix – Illustration of STRIPS Transformer Computations

In this appendix, we illustrate the operations of the STRIPS Transformer (see Eqs. 11-18), using the same traces as Figure 1, belonging to the `simple` domain. We use the optimal parameterization θ^* obtained from the ground-truth domain $M = \texttt{simple}$, and set $Q_{p_l}, K_{p_l}, V_{p_l}$ using Eqs. 8-10. Figure 3 shows the attention scores computed by each head, the application of strict future masking with stick-breaking attention, and the resulting output vectors y_{p_l}. It can be observed that the transformer correctly predicts that the first trace (Fig. 3(a)) is positive and the second trace (Fig. 3(b)) is negative.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74: 1599–1686.

Bonet, B.; and Geffner, H. 2020. Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325, 2322–2329. IOS Press.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Gösgens, J.; Jansen, N.; and Geffner, H. 2024. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. *arXiv preprint arXiv:2411.14995*.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339: 104256.

Li, K.; Hopkins, A. K.; Bau, D.; Viégas, F.; Pfister, H.; and Wattenberg, M. 2023. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task. In *The Eleventh International Conference on Learning Representations*.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32(1): 231–240.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, 2980–2988.

Liu, L.; Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; and Han, J. 2019. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Rodriguez, I. D.; Bonet, B.; Romero, J.; and Geffner, H. 2021. Learning First-Order Representations for Planning from Black Box States: New Results. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, 539–548.

Tan, S.; Yang, S.; Courville, A.; Panda, R.; and Shen, Y. 2025. Scaling Stick-Breaking Attention: An Efficient Implementation and In-depth Study. In *The Thirteenth International Conference on Learning Representations*.

Vafa, K.; Chen, J. Y.; Rambachan, A.; Kleinberg, J.; and Mullainathan, S. 2024a. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37: 26941–26975.

Vafa, K.; Chen, J. Y.; Rambachan, A.; Kleinberg, J.; and Mullainathan, S. 2024b. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37: 26941–26975.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2021. Thinking like transformers. In *International Conference on Machine Learning*, 11080–11090. PMLR.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-Symbolic Learning of Lifted Action Models from Visual Traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

Yang, A.; Chiang, D.; and Angluin, D. 2024. Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yildirim, I.; and Paul, L. 2024. From task structures to world models: what do LLMs know? *Trends in Cognitive Sciences*, 28(5): 404–415.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2444–2450. AAAI Press.

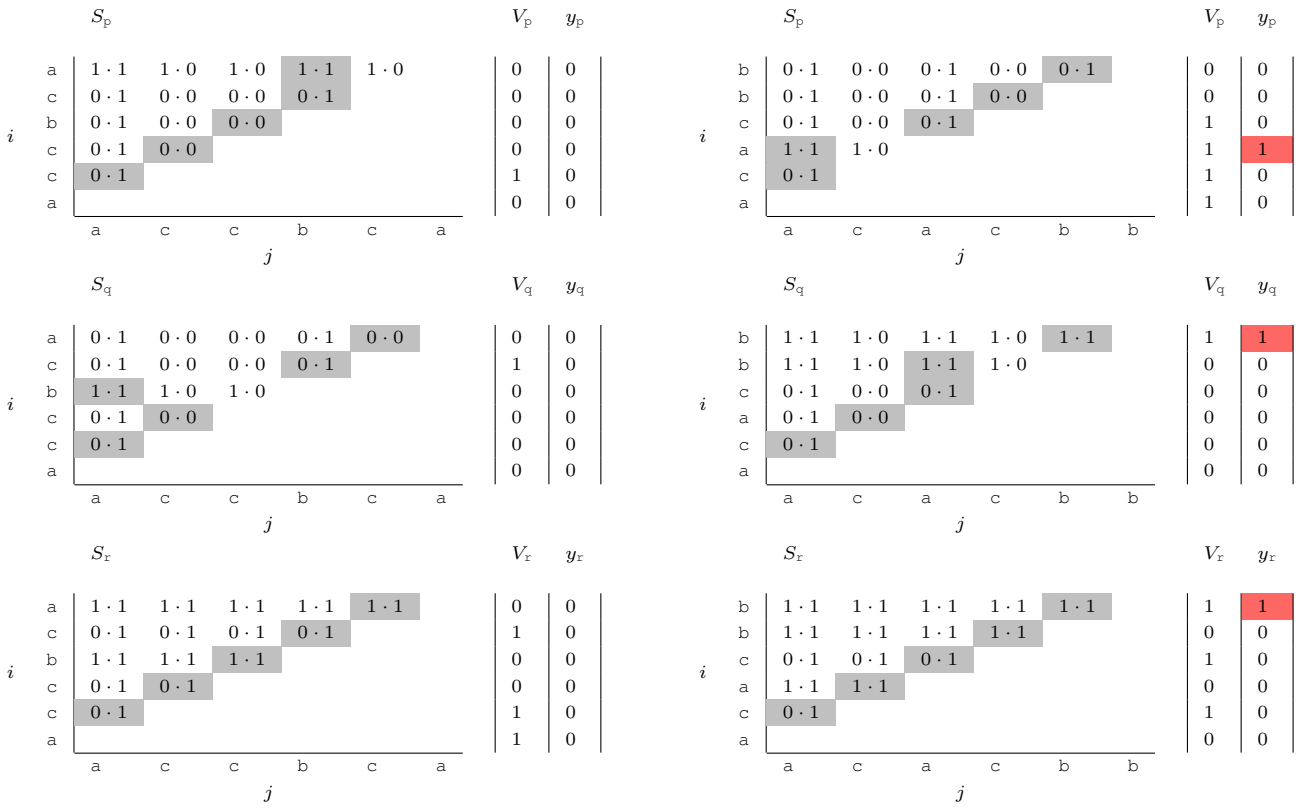

Figure 3: Self-attention computations for (a) the valid trace $(\mathtt{a}, \mathtt{c}, \mathtt{c}, \mathtt{b}, \mathtt{c}, \mathtt{a})$ and (b) the invalid trace $(\mathtt{a}, \mathtt{c}, \mathtt{a}, \mathtt{c}, \mathtt{b}, \mathtt{b})$ in the STRIPS domain `simple` (see Figure 1) using the optimal parameters θ^*. Each attention head att_{p_l}, where $p_l \in \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$ shows the score matrix $S_{p_l} = Q_{p_l} \cdot K_{p_l}^\top$ and the strict future masking operation (blank cells are masked). Since every score $S_{p_l}(i, j)$ is either 0 or 1, after applying the stick-breaking normalization each row i contains a single normalized score $S'_{p_l}(i, j)$ equal to one (highlighted in gray), whereas the rest are set to zero (non-gray cells). Therefore, when scores are 0 or 1, stick-breaking attention is equivalent to hard attention. Finally, we show the output $y_{p_l} = S'_{p_l} \cdot V_{p_l}$ of each attention head, where $y_{p_l}(i) = 0$ if a_i is applicable according to p_l and otherwise $y_{p_l}(i) = 1$. For the invalid trace $(\mathtt{a}, \mathtt{c}, \mathtt{a}, \mathtt{c}, \mathtt{b}, \mathtt{b})$ in (b) we can see that $y_{\mathtt{p}}(3) = 1$, meaning that $a_3 = \mathtt{a}$ is inapplicable according to $\mathtt{p}$; also, we observe that $y_{\mathtt{q}}(6) = y_{\mathtt{r}}(6) = 1$, meaning that $a_6 = \mathtt{b}$ is inapplicable according to both $\mathtt{q}$ and $\mathtt{r}$. Therefore, the trace is negative.