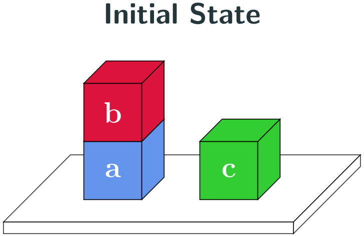
Symmetry-Aware Transformer Training for Automated Planning

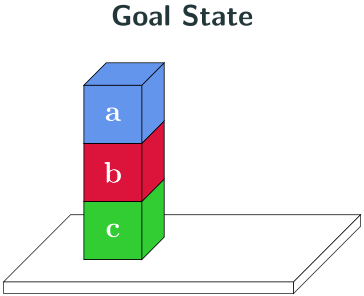
Markus Fritzsche Elliot Gestrin Jendrik Seipp

Linköping University

AAAI 2026, Singapore

The Task - Classical Planning

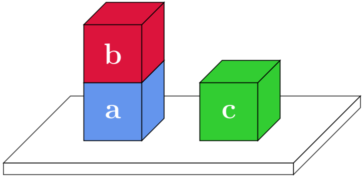




The Task - Classical Planning

The Task - Classical Planning

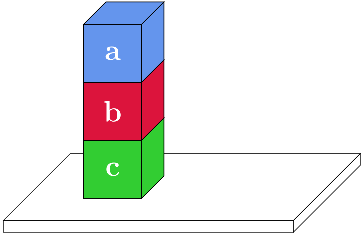
Initial State



on-table(a)
on(b, a)
on-table(c)
clear(b)
clear(c)

→
plan

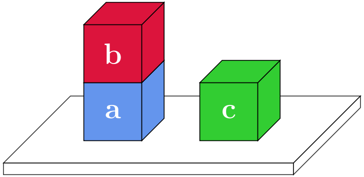
Goal State



on(b, c)
on(a, b)

The Task - Classical Planning

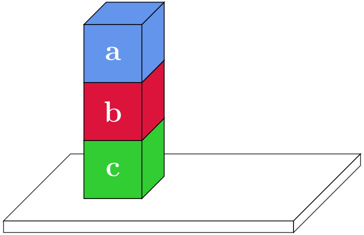
Initial State



on-table(a)
on(b, a)
on-table(c)
clear(b)
clear(c)

$\longrightarrow$
plan

Goal State



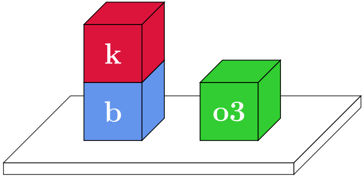
on(b, c)
on(a, b)

---

unstack(b, a) $\rightarrow$ stack(b, c) $\rightarrow$ pickup(a) $\rightarrow$ stack(a, b)

Renamed Task - Object names serve only as identifiers!

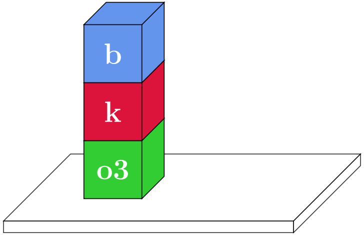
Initial State



on-table(b)
on(k, b)
on-table(o3)
clear(k)
clear(o3)

$\longrightarrow$
plan

Goal State



on(k, o3)
on(b, k)

---

unstack(k, b) $\rightarrow$ stack(k, o3) $\rightarrow$ pickup(b) $\rightarrow$ stack(b, k)

Moivation – Why Learning for Planning?

- Classical planning is fully observable, deterministic, and discrete.
- However, classical planning is **PSPACE-complete** (computationally hard).
- Traditional planners do not exploit domain knowledge.
- **Learning-based planners** exploit domain structure.
- We focus on extrapolation to larger problem sizes.

PlanGPT: Tokenization

- PlanGPT: Current SoTA Transformer-decoder planner[1]
- Still lags behind GNN-based methods.

[1] Rossetti, Nicholas, et al. "Learning general policies for planning through GPT models." Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 34, 2024.

PlanGPT: Tokenization

- PlanGPT: Current SoTA Transformer-decoder planner[1]
- Still lags behind GNN-based methods.
- Tokenization:
  - Map object names (e.g., `block1`) to abstract names o_i (random per instance).
  - This implies an upper limit of objects (e.g., 10) due to fixed vocabulary.
  - Tokenize using segmentation (split predicates/atoms into tokens).

[1] Rossetti, Nicholas, et al. "Learning general policies for planning through GPT models." Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 34, 2024.

PlanGPT: Tokenization

- PlanGPT: Current SoTA Transformer-decoder planner[1]
- Still lags behind GNN-based methods.
- Tokenization:
  - Map object names (e.g., `block1`) to abstract names o_i (random per instance).
  - This implies an upper limit of objects (e.g., 10) due to fixed vocabulary.
  - Tokenize using segmentation (split predicates/atoms into tokens).

Example mapping: `a` $\rightarrow$ `o1`, `b` $\rightarrow$ `o2`, `c` $\rightarrow$ `o3`

Tokenized sequence: `<state>, on, o1, o2, clear, o1, on-table, o2, <goal>, on, o2, o1`

[1] Rossetti, Nicholas, et al. "Learning general policies for planning through GPT models." Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 34, 2024.

Limitations of PlanGPT

- Combinatorial explosion due to symmetries, and chosen positional encodings are unsuitable for generalization.
- Four critical limitations:
  1. **Object Assignment Equivariance.** Object names are arbitrary. With $|\mathcal{O}|$ objects and $|\mathcal{V}|$ vocabulary names: $\frac{|\mathcal{V}|!}{(|\mathcal{V}|-|\mathcal{O}|)!}$ equivalent assignments
  2. **Information Leakage.** Semantic names (e.g., `loc-x1-y2`) are not randomized, which allows memorization rather than generalization.
  3. **Atom Order Equivariance.** $|\mathcal{I}|! \cdot |\mathcal{G}|!$ equivalent orderings per assignment.
  4. **Learned positional encodings.** They prevent generalization to unseen positions.

Object Assignment Equivariance

Same problem, different assignments

Mapping A

`a → o1, b → o2, c → o3`

State: `on(o1,o2), clear(o1), on-table(o2)`

Mapping B

`a → o4, b → o1, c → o7`

State: `on(o4,o1), clear(o4), on-table(o1)`

Object Assignment Equivariance

Same problem, different assignments

Mapping A

`a` $\rightarrow$ `o1`, `b` $\rightarrow$ `o2`, `c` $\rightarrow$ `o3`

State: `on(o1,o2)`, `clear(o1)`, `on-table(o2)`

Mapping B

`a` $\rightarrow$ `o4`, `b` $\rightarrow$ `o1`, `c` $\rightarrow$ `o7`

State: `on(o4,o1)`, `clear(o4)`, `on-table(o1)`

Combinatorial implication

With $|\mathcal{O}|$ objects and $|\mathcal{V}|$ vocabulary names: $\frac{|\mathcal{V}|!}{(|\mathcal{V}| - |\mathcal{O}|)!}$ equivalent assignments.

For $|\mathcal{V}| = 10$ and $|\mathcal{O}| = 3$: $\frac{10!}{(10-3)!} = 10 \times 9 \times 8 = \mathbf{720}$.

Atom Order Equivariance

States and goals are sets of atoms — order doesn't matter

Ordering A

`<state>, on, o1, o2, clear, o1, on-table, o2, <goal>, ...`

Ordering B

`<state>, clear, o1, on-table, o2, on, o1, o2, <goal>, ...`

Atom Order Equivariance

States and goals are sets of atoms — order doesn't matter

Ordering A

`<state>, on, o1, o2, clear, o1, on-table, o2, <goal>, ...`

Ordering B

`<state>, clear, o1, on-table, o2, on, o1, o2, <goal>, ...`

Combinatorial implication

With $|\mathcal{I}|$ initial atoms and $|\mathcal{G}|$ goal atoms there are $|\mathcal{I}|! \cdot |\mathcal{G}|!$ equivalent orderings per assignment.

Example: $|\mathcal{I}| = 3, |\mathcal{G}| = 1 \Rightarrow 3! \cdot 1! = \mathbf{9}$ equivalent orderings.

Symmetry-Aware Training — Architecture

Overview: We account for language-induced symmetries via architecture and training objective.

- **Atom tokens:** one token per atom (e.g., `on(o1,o2)`).
- **Goal tokens:** dedicated goal predicates (`goal-on(...)`).
- **Drop learned positional encodings (NoPE):** This makes atom ordering irrelevant and allows generalization to longer sequences.
- **Contrastive loss:** train on paired sequences with different object assignments to align representations.

Symmetry-Aware Training — Architecture

Overview: We account for language-induced symmetries via architecture and training objective.

- **Atom tokens:** one token per atom (e.g., `on(o1,o2)`).
- **Goal tokens:** dedicated goal predicates (`goal-on(...)`).
- **Drop learned positional encodings (NoPE):** This makes atom ordering irrelevant and allows generalization to longer sequences.
- **Contrastive loss:** train on paired sequences with different object assignments to align representations.

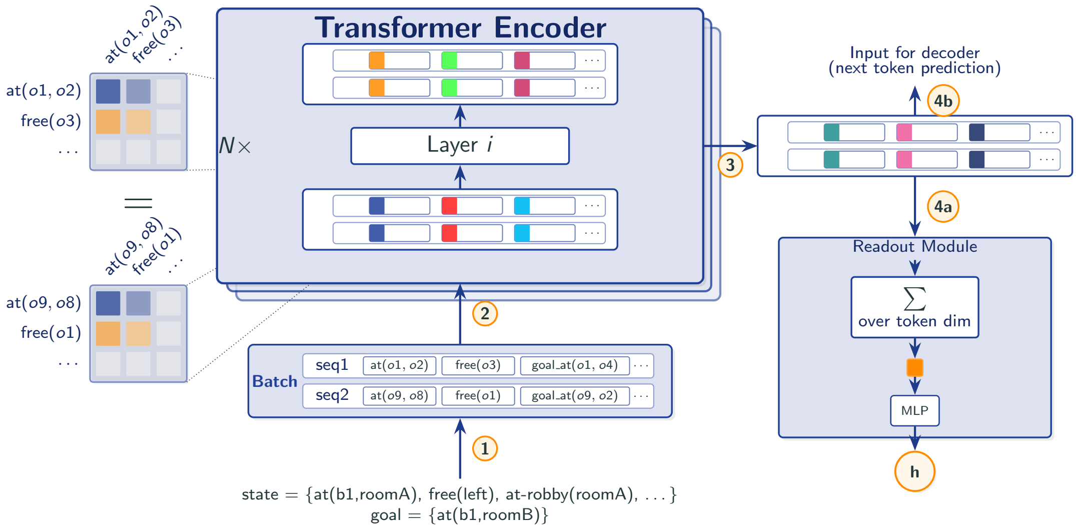
Example:

PlanGPT: `<state>, on, o1, o2, ..., <goal>, on, o2, o1`

vs

Ours: `on(o1, o2), ..., goal-on(o2, o1)`

Encoder with Contrastive Objective



SymT$^{\mathrm{E}}$

Encoder-only

Estimates goal distance; used as a greedy heuristic policy.

$$\pi_\theta(s) = \arg\min_{a \in \mathcal{A}(s)} h_\theta\big(\mathrm{Tok}\big(\mathrm{succ}(s, a), G\big)\big)$$

where h_θ is the trained encoder.

Symmetry-Aware Training — Architectures

SymT$^{\mathrm{E}}$

Encoder-only

Estimates goal distance; used as a greedy heuristic policy.

$$\pi_\theta(s) = \arg\min_{a \in \mathcal{A}(s)} h_\theta\big(\mathrm{Tok}\big(\mathrm{succ}(s, a), G\big)\big)$$

where h_θ is the trained encoder.

SymT$^{\mathrm{ED}}$

Encoder–Decoder

Autoregressively selects next plan token (greedy).

$$\pi_t = \arg\max_{w \in \mathcal{V}} P_\psi\big(w \mid \pi_{<t}, E_\phi(\mathrm{Tok}(s, G))\big)$$

with E_ϕ the trained encoder and P_ψ the trained decoder.

Experiments

		PlanGPT - Decoder (baseline)			SymT^{E} (ours)	SymT^{ED} (ours)		
		greedy	applicable	regrounding	greedy	greedy	applicable	regrounding
Blocks	validation	.00±.00	.00±.00	.00±.00	**1.00±.00**	**1.00±.00**	**1.00±.00**	.00±.00
	interpolation	.56±.16	.56±.16	.00±.00	**1.00±.00**	**1.00±.00**	**1.00±.00**	**1.00±.00**
	extrapolation	.00±.00	.00±.00	.00±.00	.05±.07	.07±.02	**.13±.05**	.00±.00
Gripper	validation	.00±.00	.00±.00	.00±.00	**1.00±.00**	.17±.24	**1.00±.00**	**1.00±.00**
	interpolation	.00±.00	.44±.16	.00±.00	.89±.16	.67±.00	**1.00±.00**	**1.00±.00**
	extrapolation	.00±.00	.00±.00	.00±.00	.02±.03	.00±.00	.15±.06	**.79±.16**
Visitall	validation	.00±.00	.14±.12	.00±.00	**1.00±.00**	.33±.09	.93±.04	.99±.02
	interpolation	.05±.04	.67±.18	.41±.22	**1.00±.00**	.87±.01	.99±.01	**1.00±.00**
	extrapolation	.00±.00	.02±.02	.00±.00	.42±.11	.00±.00	.15±.05	**.64±.12**
Logistics	validation	.00±.00	**.08±.12**	.00±.00	.00±.00	.00±.00	.00±.00	.00±.00
	interpolation	.07±.05	**.44±.09**	.19±.14	.11±.00	.22±.31	.26±.29	.22±.31
	extrapolation	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**

Summary

- Adapting Transformers for symmetries
- Architecture-guaranteed partial equivariance
- Loss-encouraged full equivariance
- Significant improvements over SoTA
- **But still lags behind GNNs in terms of generalization!**

Thank you for your attention!

Appendix — Decoder