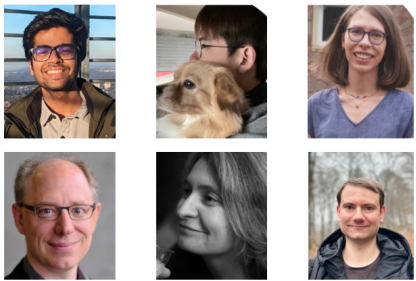
On the Ability of Transformers to Verify Plans

Yash Sarrof[†,1], Yupei Du[*,1], Katharina Stein[*,1],
Alexander Koller[1], Sylvie Thiébaux[2,3], Michael Hahn[1]

[1]Saarland University [2]Australian National University [3]University of Toulouse / LAAS-CNRS

[†]Led Theory [*]Led Experiments



Prior Work: Inconsistent Transformer Success

- Transformer-based LLMs have had mixed results in planning tasks.[1,2]
- They can perform well in distribution, but on longer plans or OOD, their performance drops.
- Why does this happen?
- We take *first steps* towards *understanding* this.

Domain	In-Distrib.		Long	
	valid. rate	*exec. rate*	*valid. rate*	*exec. rate*
BLOCKS	98.5%	98.5%	13.5%	23.5%
LOGISTICS	**100%**	**100%**	14.0%	20.5%
BARMAN	**100%**	**100%**	25.0%	43.5%
CHILDSNACK	**100%**	**100%**	66.0%	67.0%
DEPOTS	98.5%	98.5%	31.0%	37.0%
DRIVERLOG	**100%**	**100%**	31.0%	50.7%
GRIPPERS	99.0%	99.0%	50.5%	76.0%
SATELLITE	99.0%	99.2%	51.5%	53.0%

Domain	Unseen	
	valid. rate	*exec. rate*
HANOI	**0%**	35%
STORAGE	**0%**	**1%**

Domain	Obfuscated	
	valid. rate	*exec. rate*
BLOCKS	**0%**	**0%**
LOGISTICS	**0%**	**0%**

Table 1: Performance of the fine-tuned LLM across various test sets with no additional strategies applied. Although the LLM performs well on in-distribution, it struggles to generalize to OOD cases.

Table from Fritzsche et al.[2]

[1] Huang et al. "Chasing Progress, Not Perfection: Revisiting Strategies for End-to-End LLM Plan Generation." ICAPS, 2025.

[2] Fritzsche et al. "Symmetry-Aware Transformer Training for Automated Planning." ICAPS, 2026.

Some Preliminaries

Planning Domain

A domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ consists of predicates and action schemas that act as templates.

- $\mathcal{P}$: templates for propositions that define the state of the world.
- $\mathcal{A}$: templates for actions (each has preconditions and effects) that change the state.

Some Preliminaries

Planning Domain

A domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ consists of predicates and action schemas that act as templates.

- $\mathcal{P}$: templates for propositions that define the state of the world.
- $\mathcal{A}$: templates for actions (each has preconditions and effects) that change the state.

Planning Instance

A planning instance instantiates a domain with concrete objects, an initial state, and a goal state.

$$\Pi = \langle \mathcal{D}, \mathcal{O}, I, G \rangle$$

Some Preliminaries

Planning Domain

A domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ consists of predicates and action schemas that act as templates.

- $\mathcal{P}$: templates for propositions that define the state of the world.
- $\mathcal{A}$: templates for actions (each has preconditions and effects) that change the state.

Planning Instance

A planning instance instantiates a domain with concrete objects, an initial state, and a goal state.

$$\Pi = \langle \mathcal{D}, \mathcal{O}, I, G \rangle$$

Plan & Validity

A plan is a sequence of actions $\pi = [a_1, \ldots, a_n]$.
It is valid if every action is applicable and starting from the initial state, our final state is the goal.

Example of a Domain

Colors Domain

Predicates:

$\mathcal{P} = \{\text{bag}(b),\ \text{color}(c),\ \text{hasColor}(b, c)\}$

Action $\text{add}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\text{hasColor}(b, c)\}$

Action $\text{remove}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\neg\, \text{hasColor}(b, c)\}$

Example of a Domain

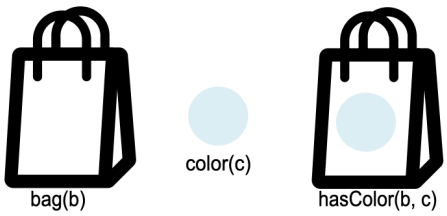
Colors Domain

Predicates:

$\mathcal{P} = \{\text{bag}(b),\ \text{color}(c),\ \text{hasColor}(b, c)\}$

Action $\text{add}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\text{hasColor}(b, c)\}$

Action $\text{remove}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\neg\, \text{hasColor}(b, c)\}$



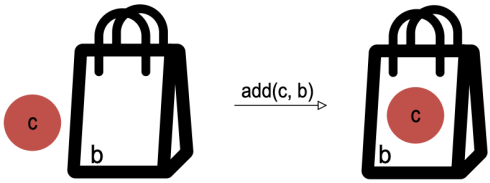
Example of a Domain

Colors Domain

Predicates:

$\mathcal{P} = \{\text{bag}(b),\ \text{color}(c),\ \text{hasColor}(b, c)\}$

Action	$\text{add}(c, b)$
Pre:	$\{\text{bag}(b),\ \text{color}(c)\}$
Eff:	$\{\text{hasColor}(b, c)\}$
Action	$\text{remove}(c, b)$
Pre:	$\{\text{bag}(b),\ \text{color}(c)\}$
Eff:	$\{\neg\, \text{hasColor}(b, c)\}$



Example of a Domain

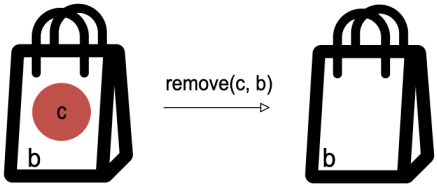
Colors Domain

Predicates:

$\mathcal{P} = \{\text{bag}(b),\ \text{color}(c),\ \text{hasColor}(b, c)\}$

Action $\text{add}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\text{hasColor}(b, c)\}$

Action $\text{remove}(c, b)$
Pre: $\{\text{bag}(b),\ \text{color}(c)\}$
Eff: $\{\neg\,\text{hasColor}(b, c)\}$



Example of an Instance

Objects = {O4, O7, O11, O13}

Initial = {bag(O4), bag(O13), color(O7), color(O11)}

Goal = {hasColor(O4, O7), hasColor(O4, O11)}

Example of a Valid Plan

Generalization: Longer Plans & More Objects

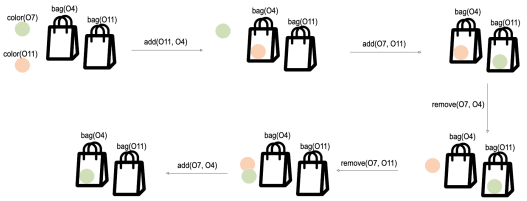
Train on small examples $\Longrightarrow$ verify harder examples

Generalization: Longer Plans & More Objects

Train on small examples $\Longrightarrow$ verify harder examples

Fixed Universe

- (I, G) change, O is fixed.



Generalization: Longer Plans & More Objects

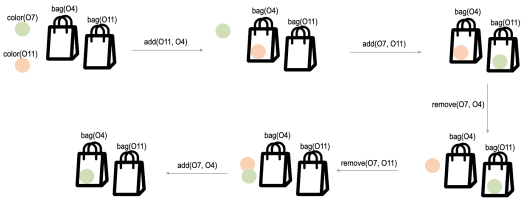
Train on small examples $\Longrightarrow$ verify harder examples

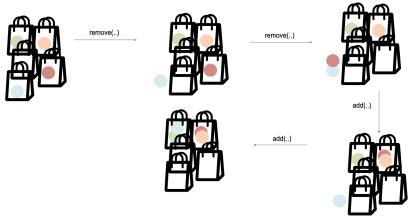
Fixed Universe

- (I, G) change, O is fixed.

Variable Universe

- (I, G, O) can change.





A framework for length generalization

C-RASP[1]

Symbolic Programming language.

[1] Yang et al. *Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers.* COLM 2024

[2] Huang et al. *A Formal Framework for Understanding Length Generalization in Transformers.*

ICLR 2025

A framework for length generalization

C-RASP[1]

Symbolic Programming language.

Boolean-Valued Operations	
Initial	$P(i) := Q_\sigma(i) \quad \text{for } \sigma \in \Sigma$
Boolean	$P(i) := \neg P_1(i)$ $P(i) := P_1(i) \wedge P_2(i)$
Positional	$P(i) := \phi(i) \text{ for } \phi \in \Phi$
Constant	$P(i) := \top$
Comparison	$P(i) := C_1(i) \leq C_2(i)$

Count-Valued Operations	
Counting	$C(i) := \#\left[j \leq i, \psi(i,j)\right] \; P(j)$ for $\psi \in \Psi \cup \{\top\}$
Conditional	$C(i) := P(i) \; ? \; C_1(i) : C_2(i)$
Add/Subtract	$C(i) := C_1(i) \pm C_2(i)$
Constant	$C(i) := 1$

[1] Yang et al. *Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers.* COLM 2024

[2] Huang et al. *A Formal Framework for Understanding Length Generalization in Transformers.* ICLR 2025

A framework for length generalization

C-RASP[1]

Symbolic Programming language.

Guarantee [2]

Transformers are predicted to length generalize on tasks for which a **C-RASP** program exists.

Boolean-Valued Operations	
Initial	$P(i) := Q_\sigma(i) \quad \text{for } \sigma \in \Sigma$
Boolean	$P(i) := \neg P_1(i)$ $P(i) := P_1(i) \wedge P_2(i)$
Positional	$P(i) := \phi(i) \text{ for } \phi \in \Phi$
Constant	$P(i) := \top$
Comparison	$P(i) := C_1(i) \leq C_2(i)$

Count-Valued Operations	
Counting	$C(i) := \#\left[j \leq i, \psi(i,j)\right] \; P(j)$ for $\psi \in \Psi \cup \{\top\}$
Conditional	$C(i) := P(i) \; ? \; C_1(i) : C_2(i)$
Add/Subtract	$C(i) := C_1(i) \pm C_2(i)$
Constant	$C(i) := 1$

[1] Yang et al. *Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers.* COLM 2024

[2] Huang et al. *A Formal Framework for Understanding Length Generalization in Transformers.* ICLR 2025

Introducing **C*-RASP**: Framework for length and symbolic generalization

Problem

C-RASP cant be used when vocabulary also increases during test time.

Introducing **C*-RASP**: Framework for length and symbolic generalization

Problem

C-RASP cant be used when vocabulary also increases during test time.

Our Contribution

C*-RASP extends **C-RASP** and helps establish formal guarantees for cases where both length and vocabulary grows during test time.

Introducing **C*-RASP**: Framework for length and symbolic generalization

Problem

C-RASP cant be used when vocabulary also increases during test time.

Our Contribution

C*-RASP extends **C-RASP** and helps establish formal guarantees for cases where both length and vocabulary grows during test time.

Boolean-Valued Operations	
Initial	$P(i) := Q_\sigma(i) \quad \text{for } \sigma \in \Sigma$
Boolean	$P(i) := \neg P_1(i)$ $P(i) := P_1(i) \wedge P_2(i)$
Positional	$P(i) := \phi(i) \text{ for } \phi \in \Phi$
Constant	$P(i) := \top$
Comparison	$P(i) := C_1(i) \leq C_2(i)$

Count-Valued Operations	
Counting	$C(i) := \#\,[j \leq i, \psi(i,j)]\ P(j)$ for $\psi \in \Psi \cup \{\top\}$
Match	$C(i) := \#\,[j \leq i, \chi(i,j)]$
Conditional	$C(i) := P(i)\ ?\ C_1(i)\ :\ C_2(i)$
Add/Subtract	$C(i) := C_1(i) \pm C_2(i)$
Constant	$C(i) := 1$

Introducing **C*-RASP**: Framework for length and symbolic generalization

Problem

C-RASP cant be used when vocabulary also increases during test time.

Our Contribution

C*-RASP extends **C-RASP** and helps establish formal guarantees for cases where both length and vocabulary grows during test time.

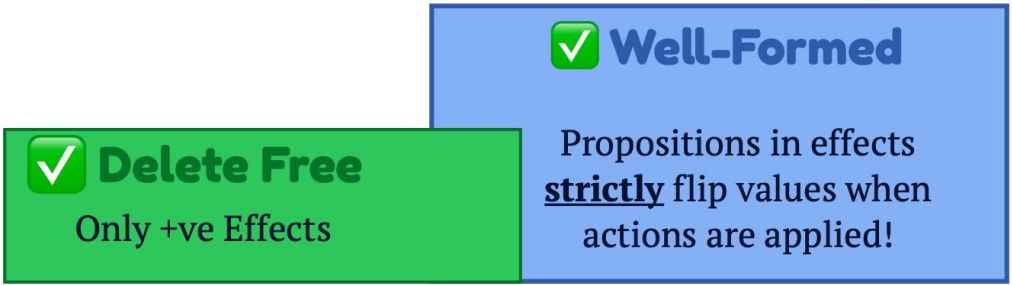
Boolean-Valued Operations	
Initial	$P(i) := Q_\sigma(i) \quad \text{for } \sigma \in \Sigma$
Boolean	$P(i) := \neg P_1(i)$ $P(i) := P_1(i) \wedge P_2(i)$
Positional	$P(i) := \phi(i) \text{ for } \phi \in \Phi$
Constant	$P(i) := \top$
Comparison	$P(i) := C_1(i) \leq C_2(i)$

Count-Valued Operations	
Counting	$C(i) := \#\,[j \leq i, \psi(i,j)]\ P(j)$ for $\psi \in \Psi \cup \{\top\}$
Match	$C(i) := \#\,[j \leq i, \chi(i,j)]$
Conditional	$C(i) := P(i)\ ?\ C_1(i)\ :\ C_2(i)$
Add/Subtract	$C(i) := C_1(i) \pm C_2(i)$
Constant	$C(i) := 1$

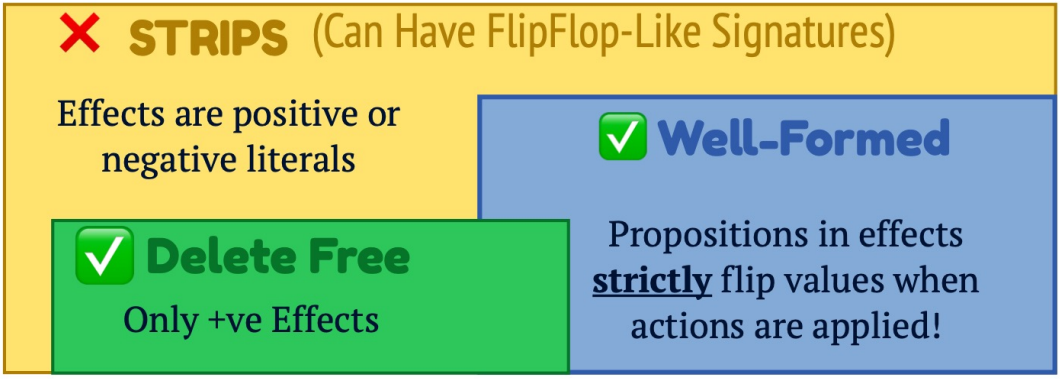
- **C*-RASP** program exists $\rightarrow$ Transformers generalize to longer lengths + bigger vocabulary.

Main Results about Planning Classes

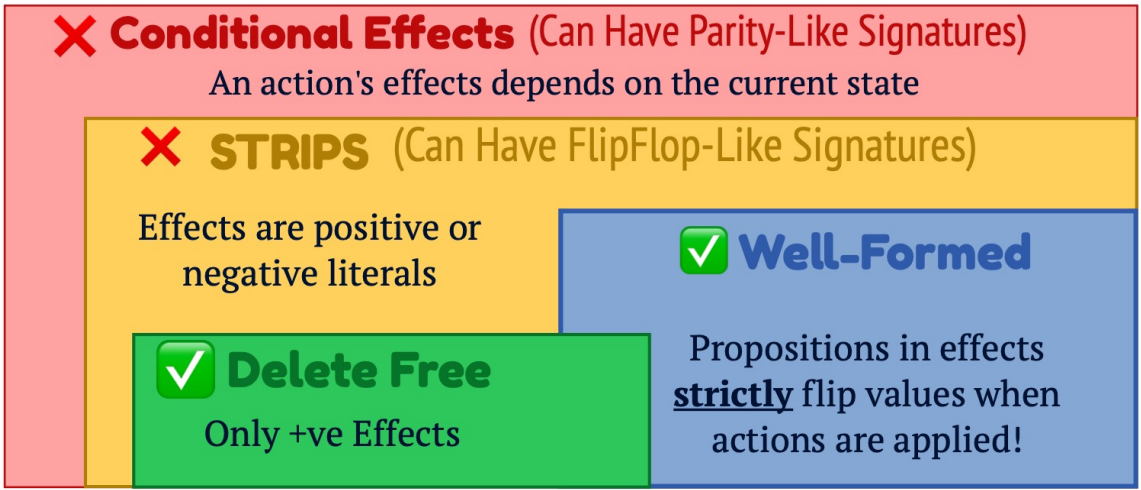
Main Results about Planning Classes



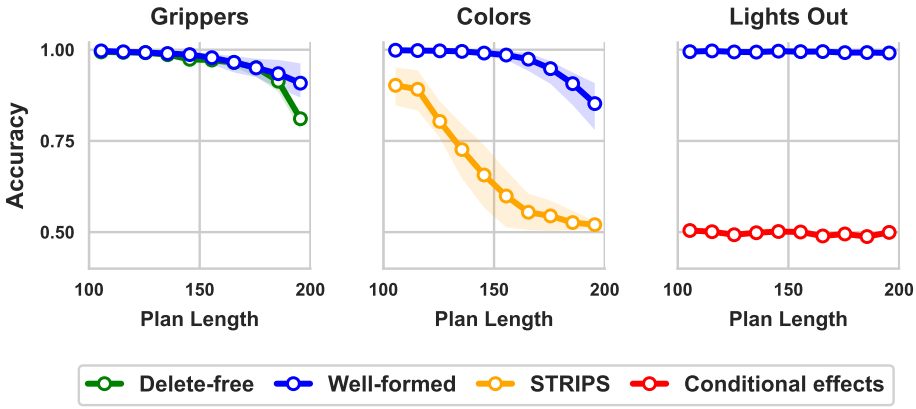
Main Results about Planning Classes



Main Results about Planning Classes



Minimal pair of experiments validate our theory!



Limitations & Takeaways

Idealized Framework

Generalization guarantees are based on idealized learning, not actual gradient descent.

Such frameworks have been empirically predictive.

Limitations & Takeaways

Idealized Framework

Generalization guarantees are based on idealized learning, not actual gradient descent.

Such frameworks have been empirically predictive.

Plan generation

Extending theory to plan generation is not trivial.

We take 1st steps, and hope it inspires future work.

Limitations & Takeaways

Idealized Framework

Generalization guarantees are based on idealized learning, not actual gradient descent.

Such frameworks have been empirically predictive.

Plan generation

Extending theory to plan generation is not trivial.

We take 1st steps, and hope it inspires future work.

Takeways

- **C*-RASP** can be used to understand generalization in planning.
  - longer plans
  - more objects
- The formulation of the planning problem can enable/inhibit transformer generalization.