Scalable Task Translation and Planning using Axiom Refactoring

Master Thesis Presentation

Benedikt Heuser @ University of Basel

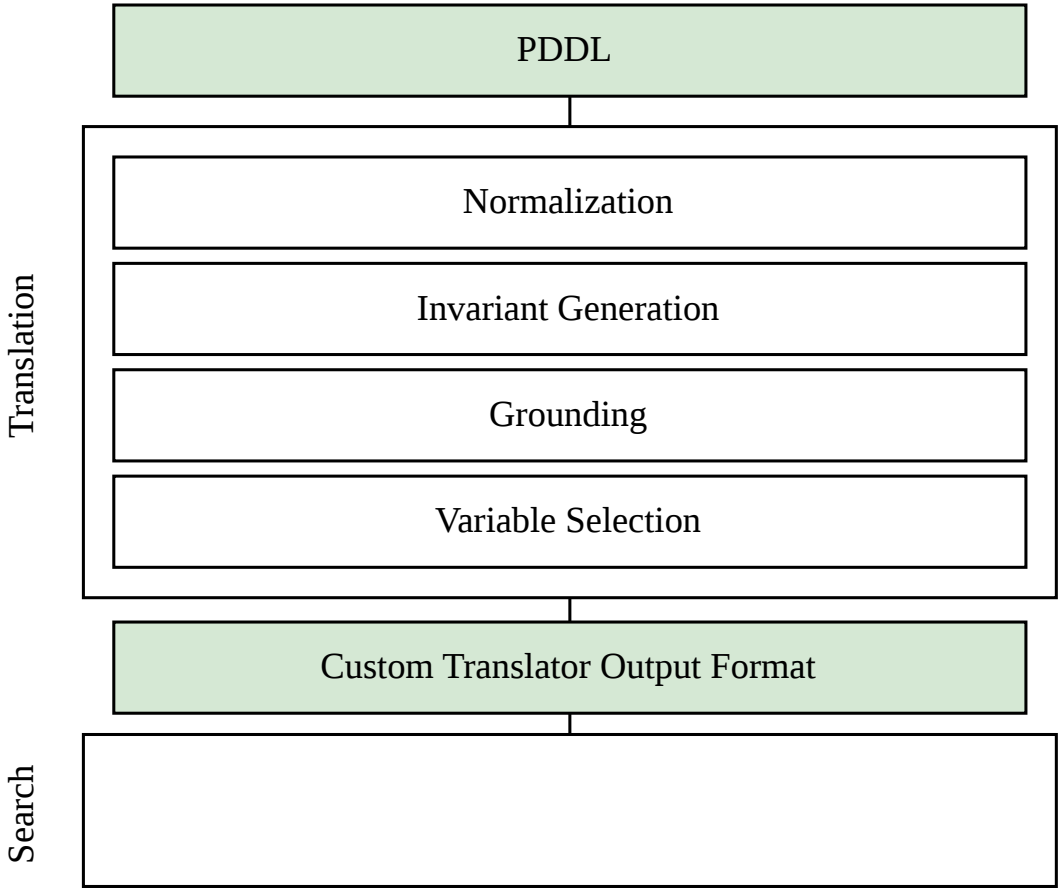
October 27, 2026

Content

1. Task Normalization in Fast Downward
2. Task Normalization using Axioms
3. Effect of Axiom Refactoring on Translation
4. Effect of Axiom Refactoring on Search
5. Memory Optimization for Search with Axioms
6. Conclusions

Task Normalization in Fast Downward

Translation Overview



DNF Normalization Example

Running Example

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle (a \vee b) \wedge (c \vee d) \wedge (e \vee f), g \rangle\}$
- $\mathcal{G} = g$

DNF Normalization Example

Running Example

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle (a \vee b) \wedge (c \vee d) \wedge (e \vee f), g \rangle\}$
- $\mathcal{G} = g$

DNF-Conversion

$$
\begin{aligned}
& (a \vee b) \wedge (c \vee d) \wedge (e \vee f) \\
\equiv\ & ((a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d)) \wedge (e \vee f) \\
\equiv\ & (a \wedge c \wedge e) \vee (a \wedge c \wedge f) \vee (a \wedge d \wedge e) \vee (a \wedge d \wedge f) \vee (b \wedge c \wedge e) \vee (b \wedge c \wedge f) \vee (b \wedge d \wedge e) \vee (b \wedge d \wedge f)
\end{aligned}
$$

DNF Normalization Example

Resulting Task

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle a \wedge c \wedge e, g \rangle, \langle a \wedge c \wedge f, g \rangle, \langle a \wedge d \wedge e, g \rangle, \langle a \wedge d \wedge f, g \rangle, \langle b \wedge c \wedge e, g \rangle, \langle b \wedge c \wedge f, g \rangle, \langle b \wedge d \wedge e, g \rangle, \langle b \wedge d \wedge f, g \rangle\}$
- $\mathcal{G} = g$

The number of operators went from 1 to 8!

Task Normalization using Axioms

Axioms

Definition: Derived Variables

A derived variable is a state variable with a binary domain. Derived variables have a default value of false and may be set to true through the evaluation of axioms.

Definition: Axioms

An axiom is a rule that consists of a *head* and a *body*. The derived variable in its head is evaluated to true for a given state s if $s \models \text{body}$.

$$\text{head} \leftarrow \text{body}$$

Axioms

Running Example

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle (a \vee b) \wedge (c \vee d) \wedge (e \vee f), g \rangle\}$
- $\mathcal{G} = g$

Axioms

Planning Task Definition with Axioms: Disjunctions

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G}, \mathcal{D}, \mathcal{A} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{D} = \{d_1, d_2, d_3\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle d_1 \wedge d_2 \wedge d_3, g \rangle\}$
- $\mathcal{G} = g$
- $\mathcal{A} =$

$$d_1 \leftarrow a \quad d_2 \leftarrow c \quad d_3 \leftarrow e$$
$$d_1 \leftarrow b \quad d_2 \leftarrow d \quad d_3 \leftarrow f$$

Axioms

Planning Task Definition with Axioms: Conjunctions

$\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G}, \mathcal{D}, \mathcal{A} \rangle$ where:

- $\mathcal{V} = \{a, b, c, d, e, f, g\}$
- $\mathcal{D} = \{d_1, d_2, d_3, d_4\}$
- $\mathcal{I} = \{a, b, c, d, e, f\}$
- $\mathcal{O} = \{\langle d_4, g \rangle\}$
- $\mathcal{G} = g$
- $\mathcal{A} =$

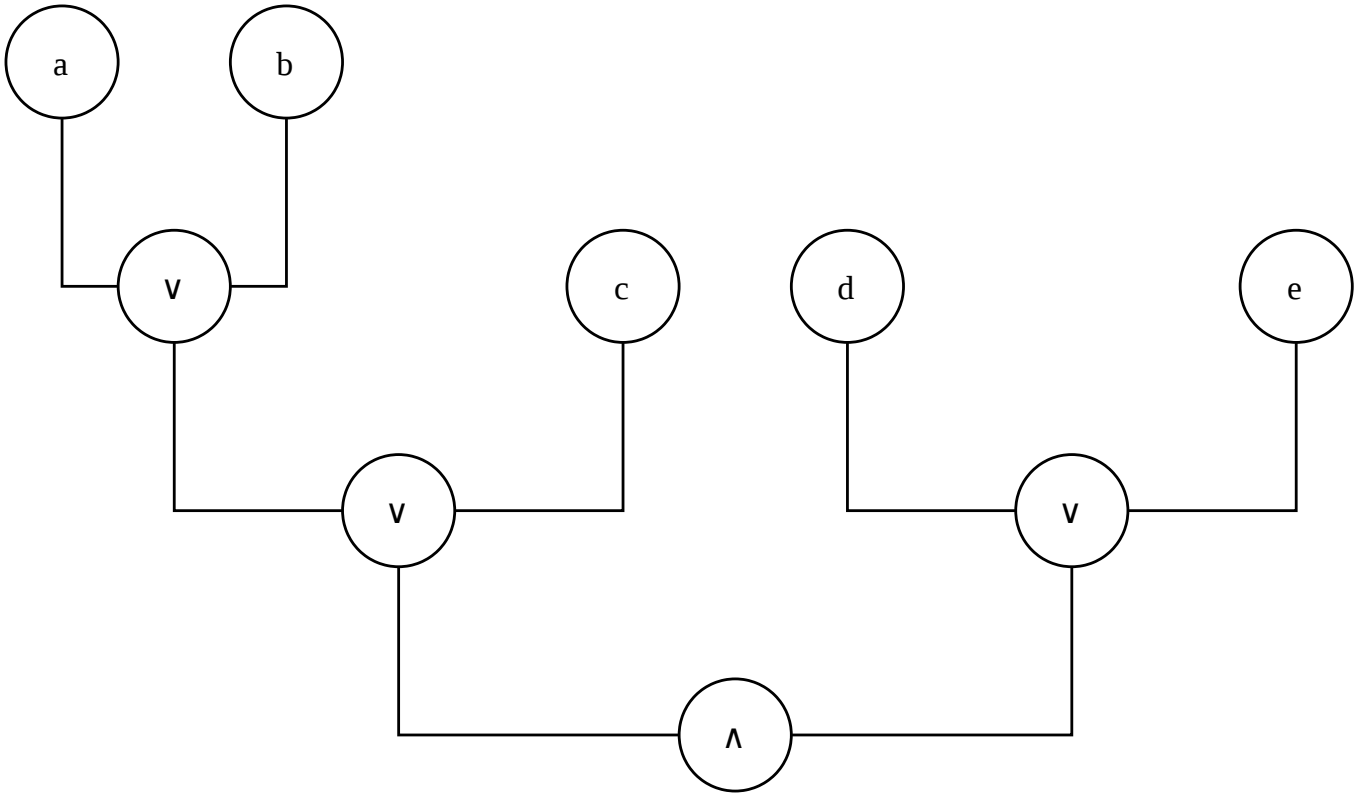
$$
\begin{array}{lll}
d_1 \leftarrow a & d_2 \leftarrow c & d_3 \leftarrow e \\
d_1 \leftarrow b & d_2 \leftarrow d & d_3 \leftarrow f \\
d_4 \leftarrow d_1 \wedge d_2 \wedge d_3 & &
\end{array}
$$

Overview of Implementations

Translator configurations

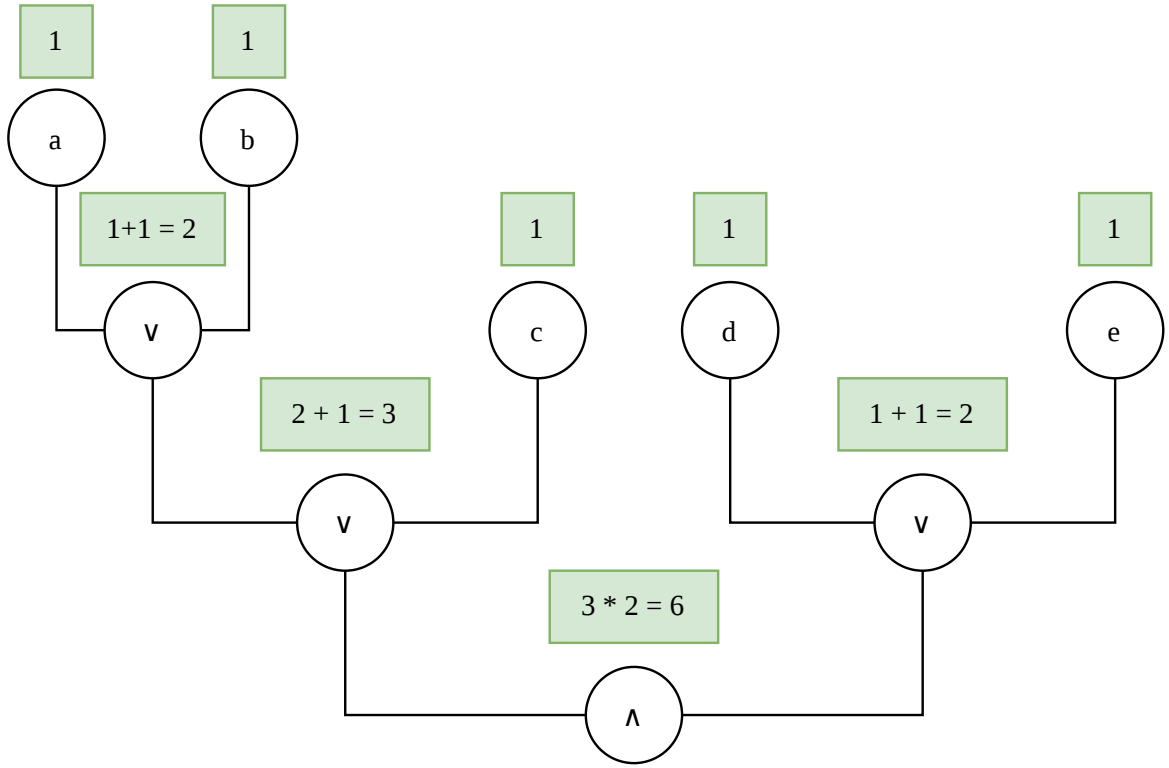
refactors	$\boldsymbol{T}^{\varnothing}$	$\boldsymbol{T}^{\{\vee\}}$	$\boldsymbol{T}^{\{\vee,\wedge\}}$	$\boldsymbol{T}^{\{\vee\}}_{\textbf{hybrid}}$
disjunctions	no	yes	yes	conditionally
conjunctions	no	no	yes	no

Blow-up Score Computation (1 / 2)



$(a \vee b \vee c) \wedge (d \vee e)$

Blow-up Score Computation (2 / 2)



$$\text{score}((a \vee b \vee c) \wedge (d \vee e)) = 6$$

Hybrid Decision Mechanism

Hybrid Strategy

For a given condition φ and a threshold τ:

$$T^{\{\vee\}}_{\text{hybrid}}(\varphi) = \begin{cases} T^{\varnothing}(\varphi) \text{ if } \text{score}(\varphi) < \tau \\ T^{\{\vee\}}(\varphi) \text{ if } \text{score}(\varphi) \geq \tau \end{cases}$$

- Only applies axiom refactoring for conditions with high scores
- τ can be chosen as a tuned parameter

Effect of Axiom Refactoring on Translation

Experimental Setup

Benchmark and limits

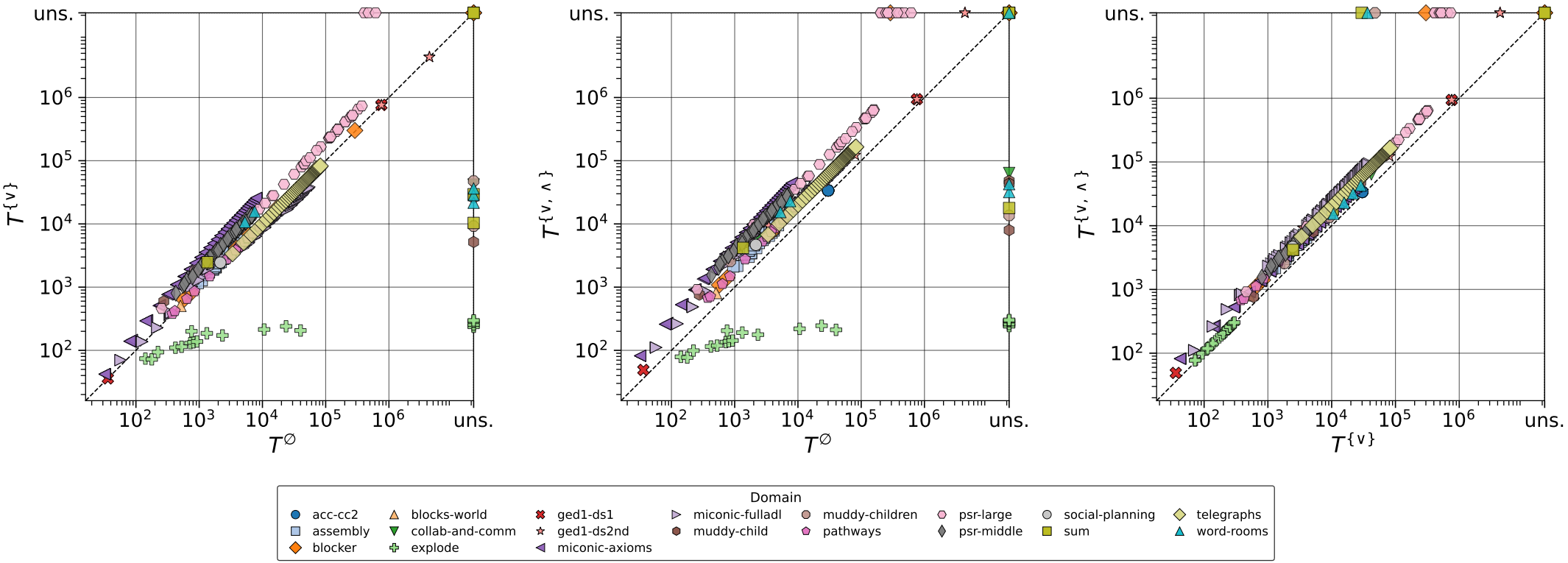
- 18 planning domains; benchmark tasks selected because they contain disjunctions in conditions
- Includes standard Fast Downward benchmarks, tasks from prior work, and generated instances of the running example
- Resource limit per run: 4 GB memory and 20 minutes
- Experiments ran on the University of Basel cluster

Translation Coverage

Successful translations across 839 tasks

Translator	**Tasks translated**
$T^{\varnothing}$ baseline	561
$T^{\{\vee\}}$ disjunctions	576
$T^{\{\vee,\wedge\}}$ disjunctions + conjunctions	561
$T^{\{\vee\}}_{\text{hybrid}}$ thresholded	579

Translator Task Representation Size



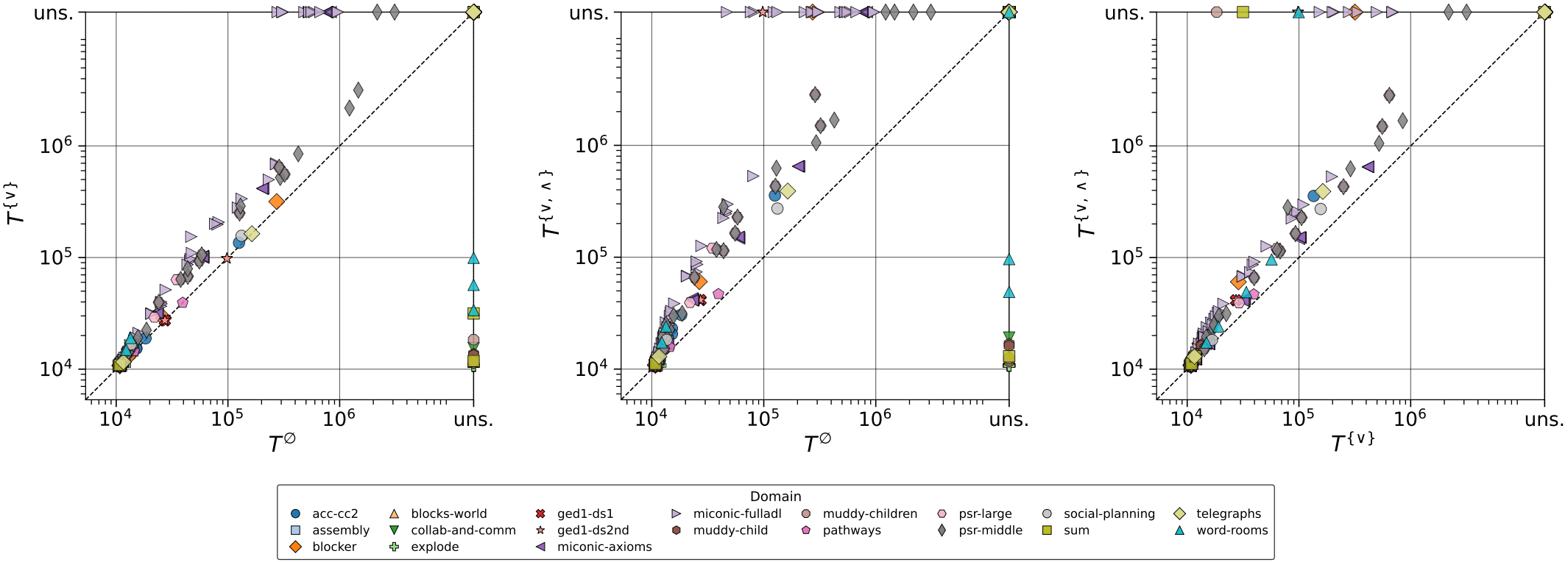
Effect of Axiom Refactoring on Search

Search Experimental Setup

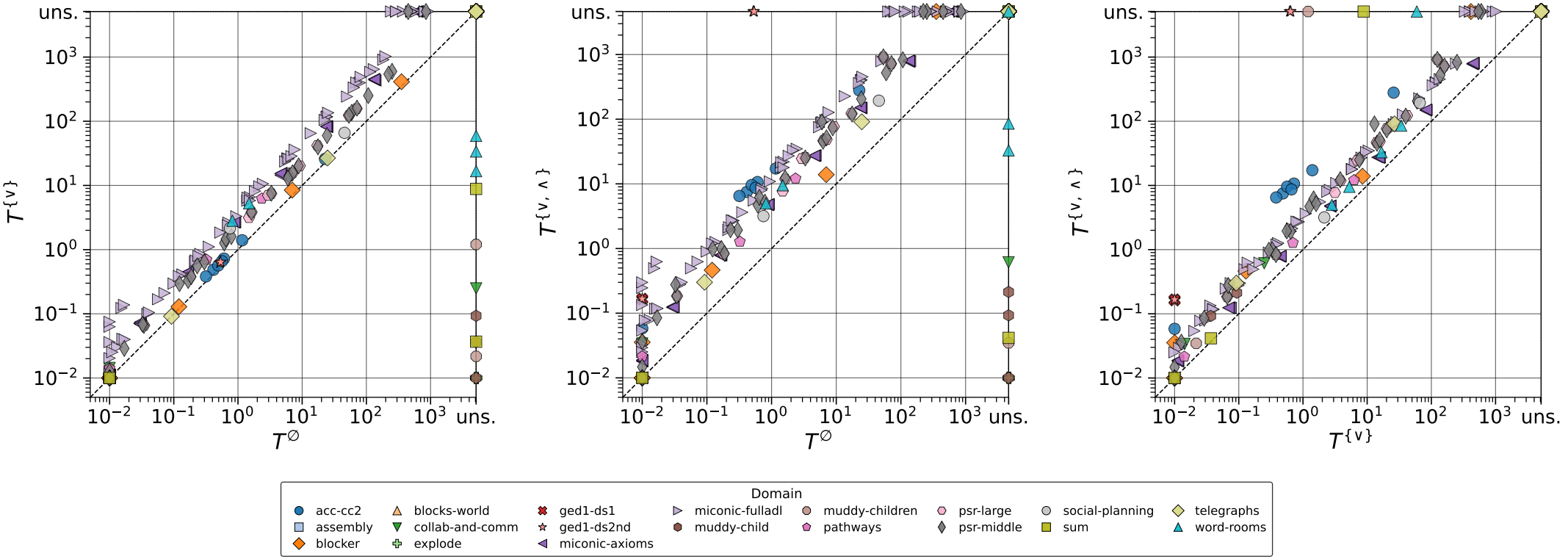
Search configurations

- Optimal planning: A^* with the blind heuristic.
- Satisficing planning: LAMA-First.

Search Memory



Search Runtime

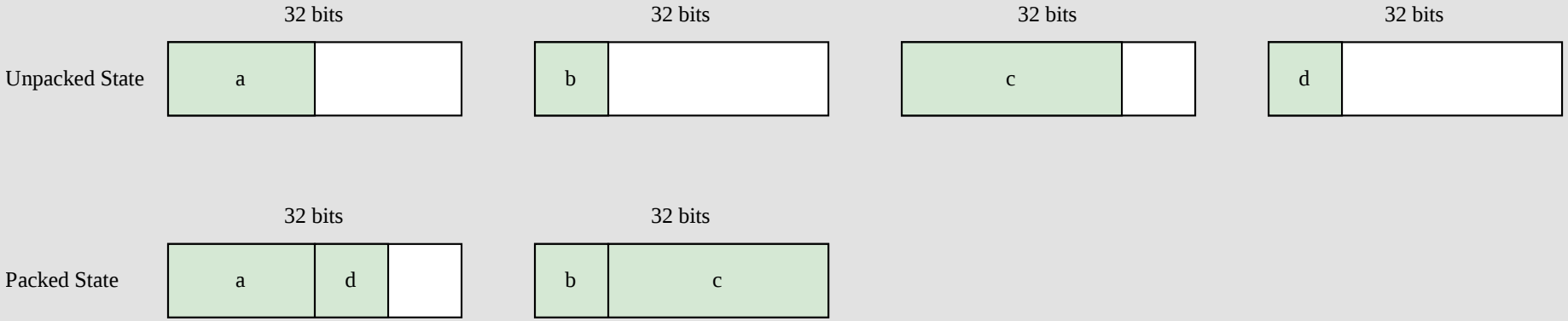


Memory Optimization for Search with Axioms

Reducing Memory Impact

State-Registry

- The state-registry persistently stores the values of generated states.
- States stored as packed states:



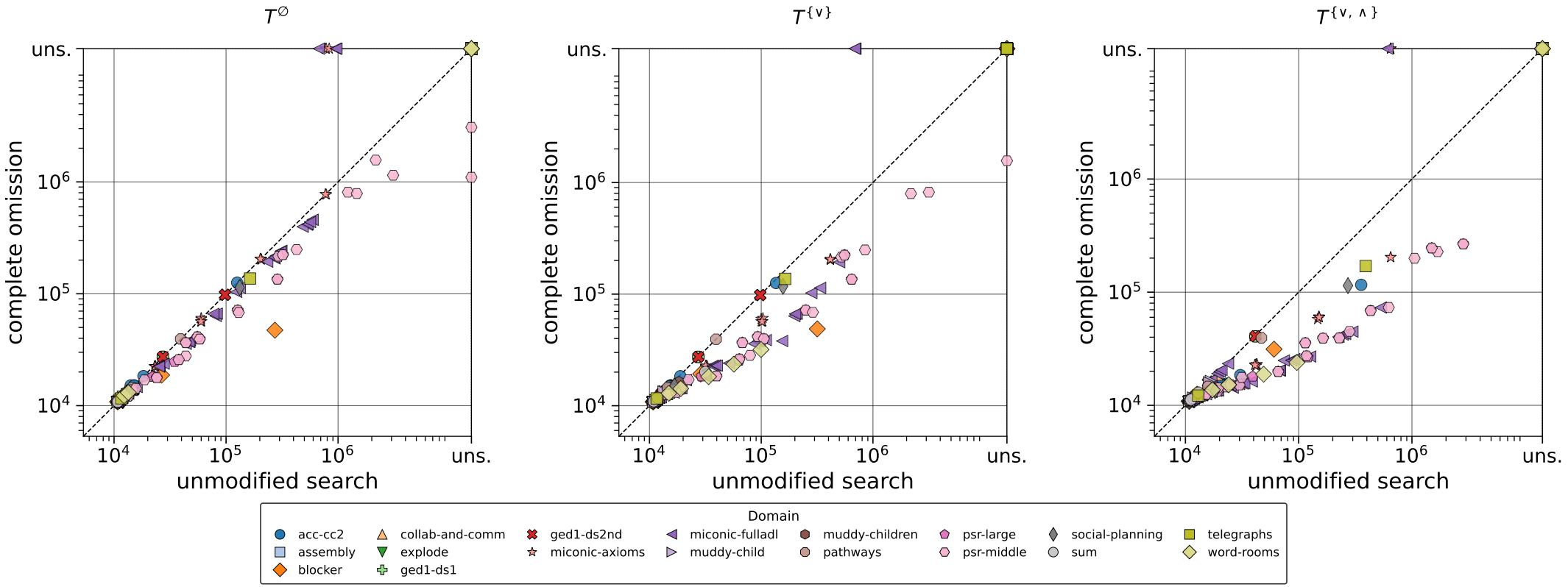
Reducing Memory Impact

Optimization Idea

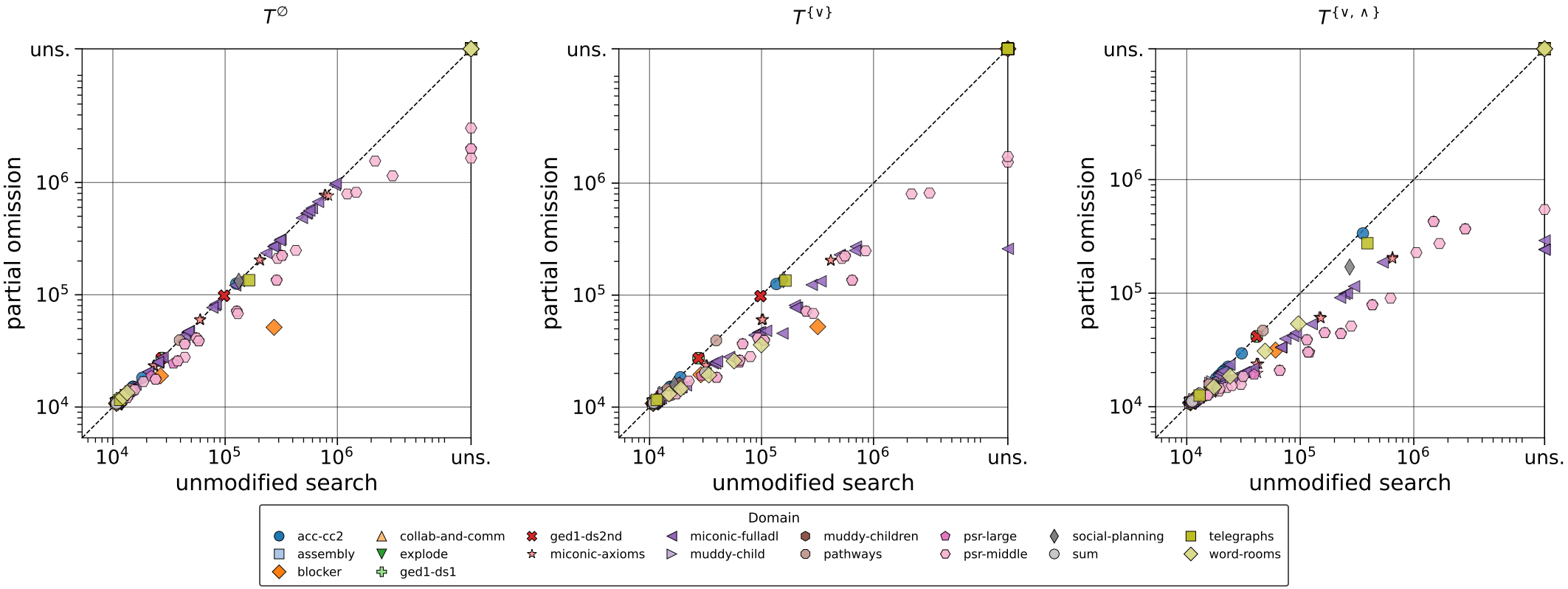
Problem: Axiom refactoring adds derived variables $\rightsquigarrow$ inflated memory use.
Idea: Store fewer variable in the state-registry.

1. Store only primary variables $\rightarrow$ **Complete axiom omission**
2. Store primary variables *and* derived variables used in goals, preconditions, or effect conditions $\rightarrow$ **Partial axiom omission**

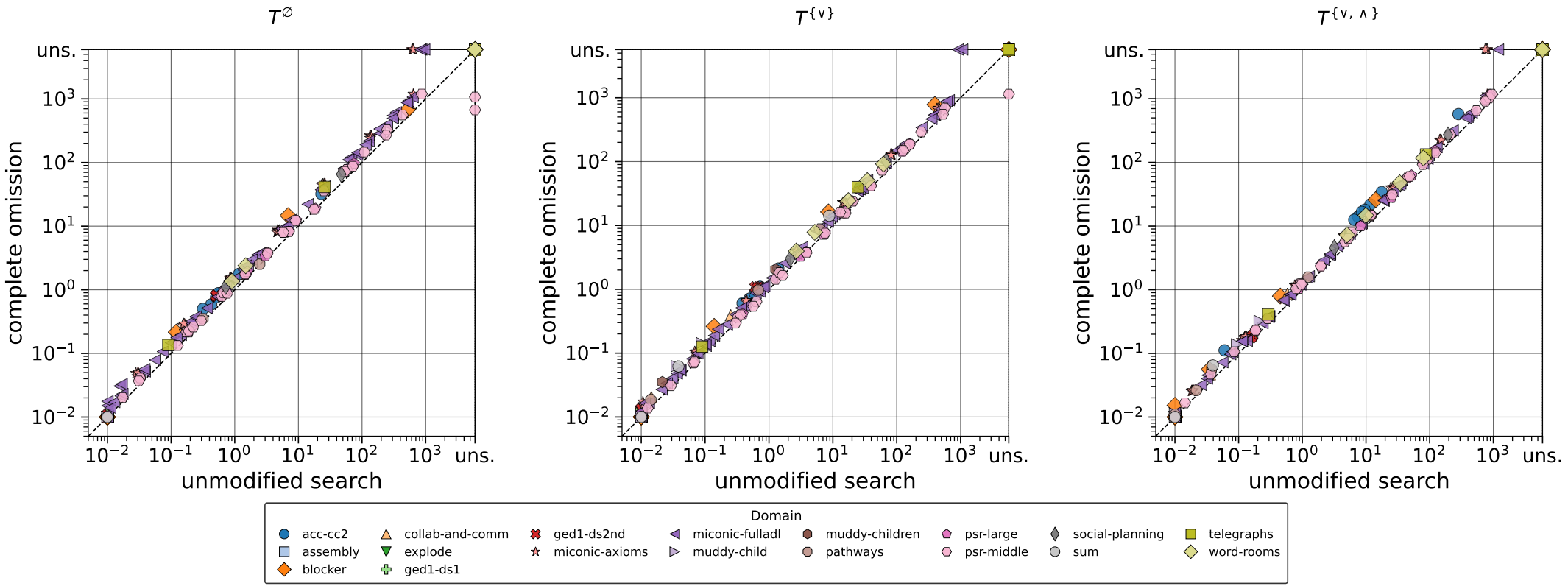
Full Axiom Omission Search Memory



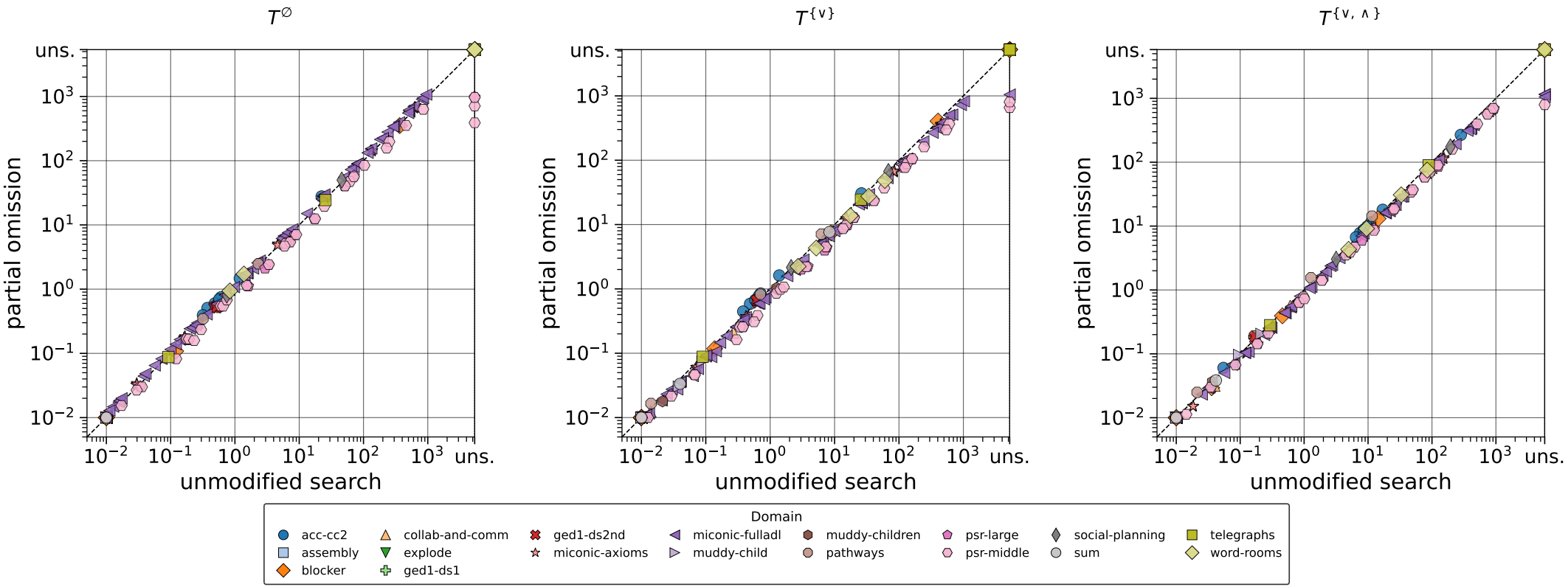
Partial Axiom Omission Search Memory



Full Axiom Omission Search Runtime



Partial Axiom Omission Search Runtime



Conclusions

Conclusions

Main findings

- Axiom refactoring allows Fast Downward to translate instances that were previously infeasible due to an exponential blow-up.
- The additional axioms generally increase runtime and memory usage during search.
- By fully or partially removing derived variables from the state-registry, we can reduce the impact on memory.

Limitations and Future Work

Limitations

- Experiments cover a limited benchmark set and limited number of search configurations
- Axiom support for admissible heuristics is low

Further Research Directions

- Develop axiom support for more heuristics
- Explore more selective, sub-formula-level hybrid refactoring
- Selective axiom evaluation
- Investigate axiom elimination and reuse strategies

Questions?

Scalable Task Translation and Planning using Axiom Refactoring

Benedikt Heuser · University of Basel