

Learning Object Priorities for Partial Grounding

Mariam Musavi Johannes K. Fichte Daniel Gnad

Linköping University, Sweden

Heidelberg University, Germany

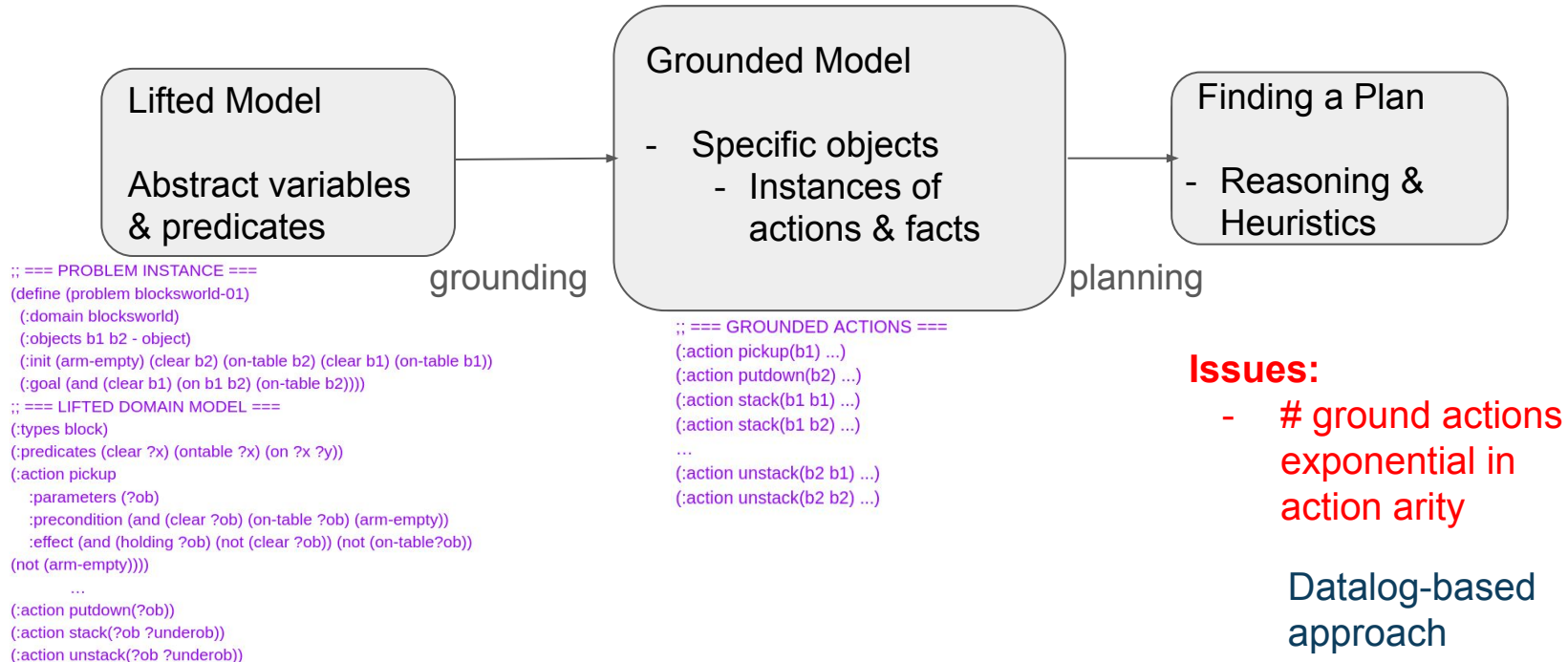


UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

RIPL Workshop, ICAPS 2026

Classical Planning Pipeline via Grounding

- Transforming first-order representation of task into grounded propositional model



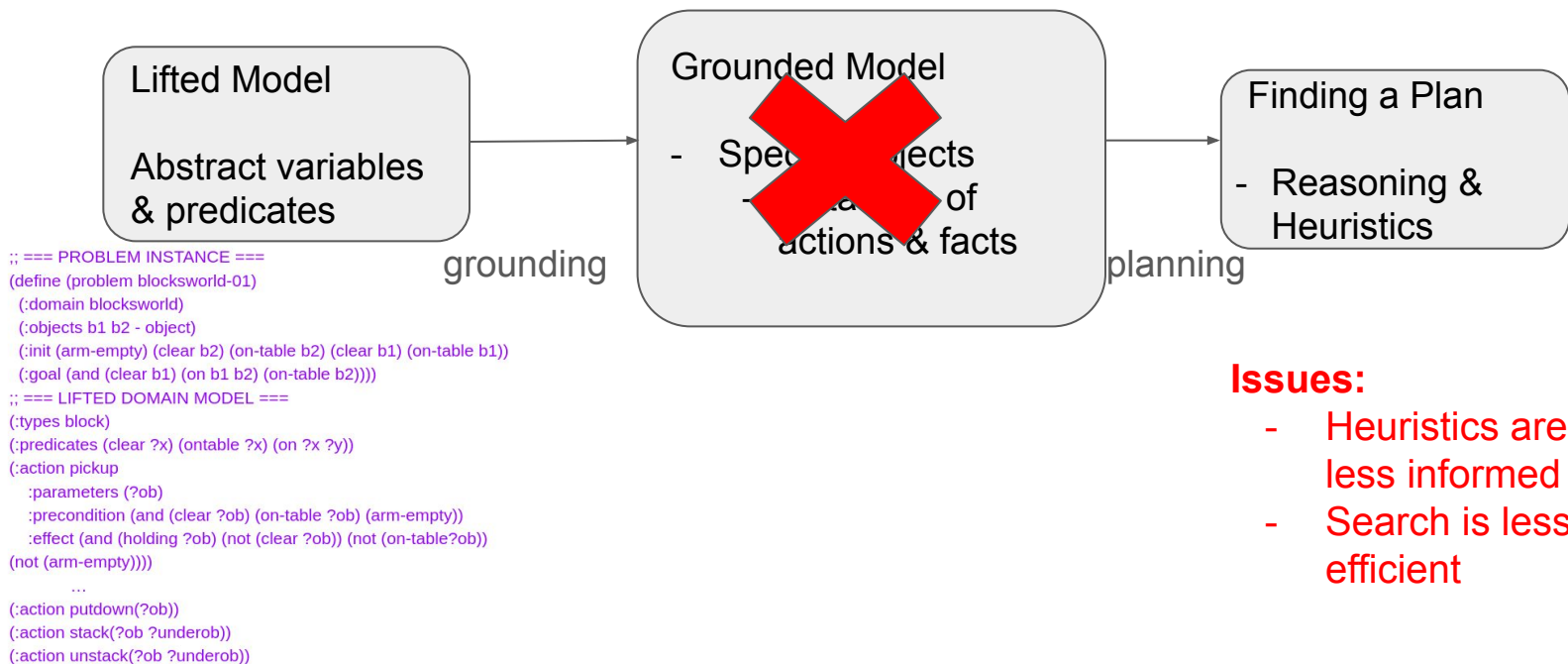
Issues:

- # ground actions exponential in action arity

Datalog-based approach

Lifted Planning

- Planning first-order representation of task



Grounding: bottlenecks

❑ Issues with Full Grounding

- ❑ **Combinatorial explosion:** action instance size  computational feasibility 
- ❑ **Memory limitations:** size of grounded model 
- ❑ **Computational costs:** reasoning and heuristic search 

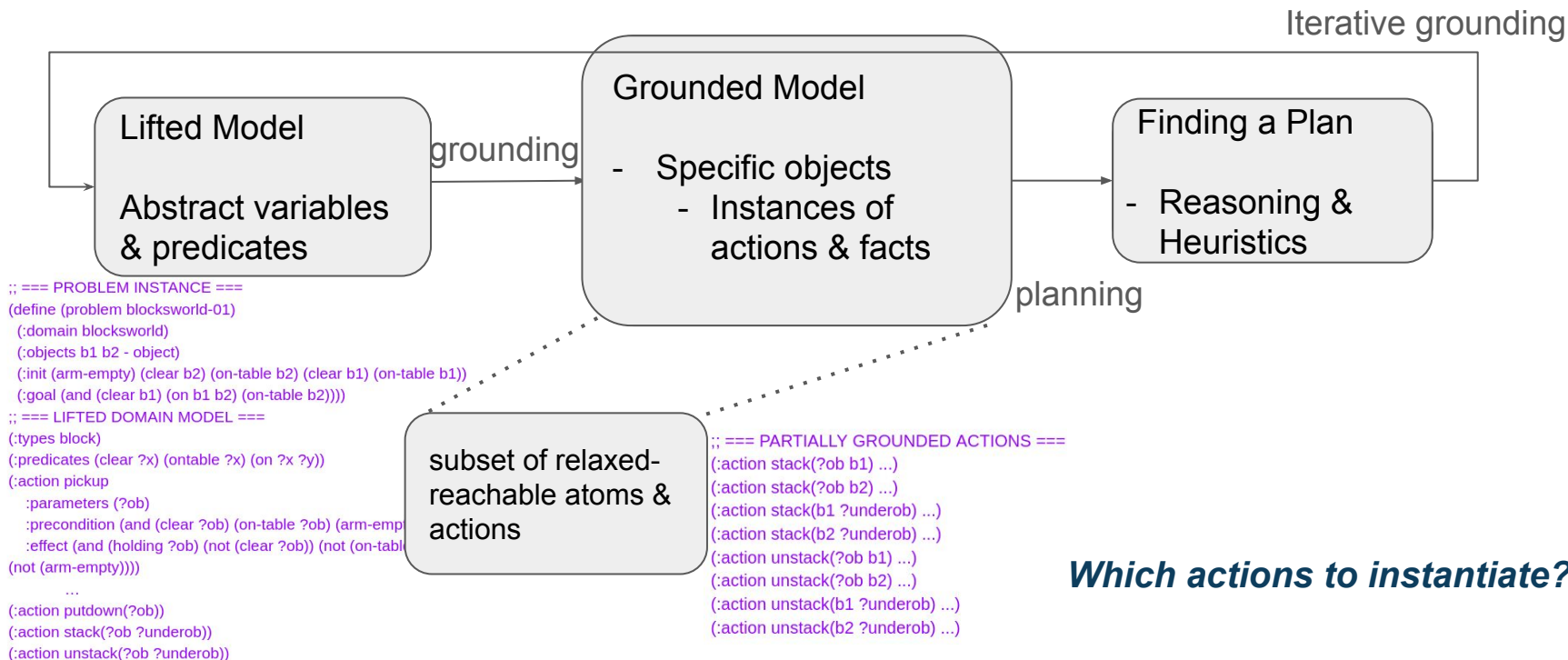
❑ Issues with Lifted Planning

- ❑ **Heuristics** are less informative 
- ❑ **Search:** less efficient 
- ❑ **Inefficient computations:** irrelevant parts of task (on lifted level) 

Partial Grounding: avoids full instantiations of ground actions & use **efficient grounded** planning techniques

Partial Grounding to the rescue!

- ❑ Transforming a lifted PDDL task into partially grounded propositional model



Which actions to instantiate?

Partial Grounding: properties

- ❑ Partial grounding Gnad et al., (2019) is **sound**, but **incomplete**
- ❑ **Stopping** early only avoids grounding some actions and state atoms
- ❑ A **plan** for the partial grounded task is a plan **valid** for the original task
- ❑ Incremental: **no plan** for partially grounded task => we can ground more => iterative approach of grounding and search

Partial Grounding: how to rank actions?

- ❑ **Action Priority Function:**
 - ❑ learn from small training instances
 - ❑ compute symbolically
- ❑ **Challenges**
 - ❑ Compute overhead
 - ❑ How to score informed priorities?

Object ordering: priority queue per action schema, each ordered by the action priority

Algorithm 1: Partial Grounding.

Input: A lifted task $\Pi = \langle \mathcal{D}, \mathcal{I} \rangle$

Output: A STRIPS task $\Pi_g = (F, O, I, G)$ Initialization

```
1  $q \leftarrow I$  ;  
2  $F \leftarrow \emptyset$  ; // Processed atoms  
3  $O \leftarrow \emptyset$  ; // Processed operators  
4 while  $\neg(q.empty() \vee G \subseteq F) \wedge \neg \text{StoppingCondition}$   
   do  
5   if  $q.containsAtom()$  then Process atoms  
6      $f \leftarrow q.popAtom()$  ;  
7      $F \leftarrow F \cup \{f\}$  ;  
8     for  $o \notin O \wedge pre(o) \subseteq F$  do  
9        $q.insert(o)$  ;  
10  else Process operators  
11     $o \leftarrow q.popHighPriorityOperator()$  ;  
12     $O \leftarrow O \cup \{o\}$  ;  
13    for  $f \notin F \wedge f \in add(o)$  do  
14       $q.insert(f)$  ;  
15 return  $(F, O, I, G)$  ;
```

Learning Per-Parameter Object Priorities

- The priority of an object o at parameter position p_i of an action schema a :

$$\rho_a(o, p_i) = \frac{|\{\hat{a} \in U_j^a \mid \hat{a}[p_i] = o\}|}{|O_j^a|}$$

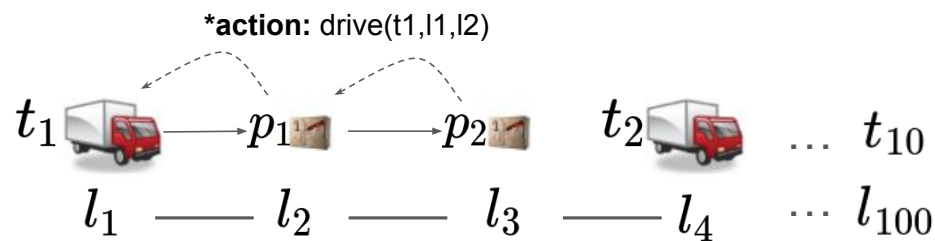
where, $\rho_a(o, p_i) \in [0, 1]$. Objects never appearing in optimal plans gets 0.

Idea: *decompose the priority calculation per-parameter slot, keeping training and inference **polynomial** instead of combinatorial.*

Example

Drive (?truck, ?l1, ?l2)

Objects: t1, t2, l1, ..., l4



Slot 1 (?truck): how likely t1 to be important?

Slot 2 (?to): how likely is l1 to be important?

Slot 3 (?from): how likely is l2 to be important?

Priority Queue Implementation

❑ **From Object to Action Priorities:** to score whole ground action $\hat{a}$ with substitution $o_i = \hat{a}[p_i]$,

1. Sum

$$\rho^{\Sigma}(\hat{a}) = \sum_{i=1}^k \rho_a(o_i, p_i)$$

Additive mitigation; a single low score doesn't completely kill an action if other parameters are highly confident

2. Product

$$\rho^{\Pi}(\hat{a}) = \prod_{i=1}^k \max(\rho_a(o_i, p_i), \epsilon)$$

Multiplicative joint probability view. Clipped at $\epsilon > 0$ to prevent complete zero-outs due to finite data gaps.

3. Optimistic

$$\rho^{opt}(\hat{a}) = \sum_{i=1}^k 1[\rho_a(o_i, p_i) > 0]$$

Binarized indicator count. Exceptionally robust to limited training coverage, focusing only on baseline viability.

Evaluation Setup

Framework: Partial grounding Gnad et al. (2019) in Fast Downward Helmert (2006)

Baseline: LAMA-first Richter and Westphal (2010)

Runtime: 10 mins

Memory limits: 4GiB

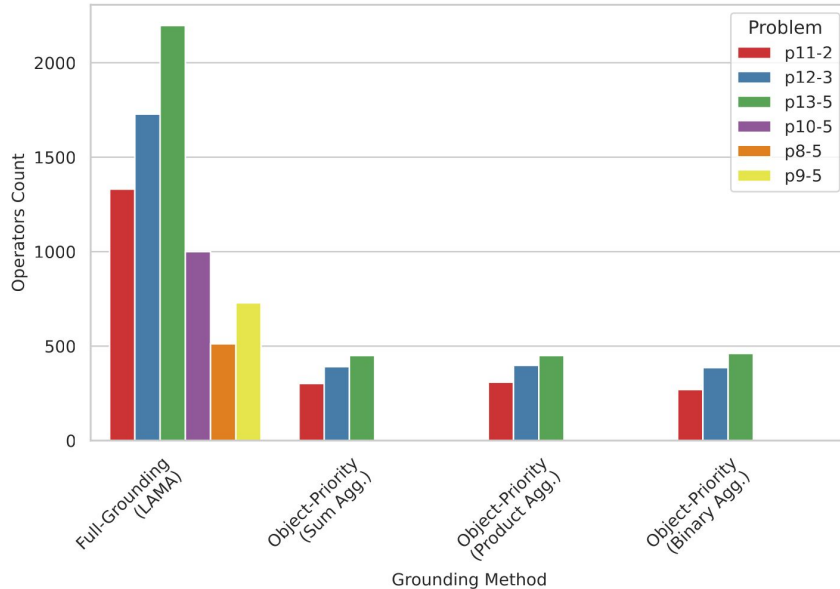
Training instances: Gnad et al., (2019)

Test instances: Hard-to-ground benchmarks Gnad et al., (2019)

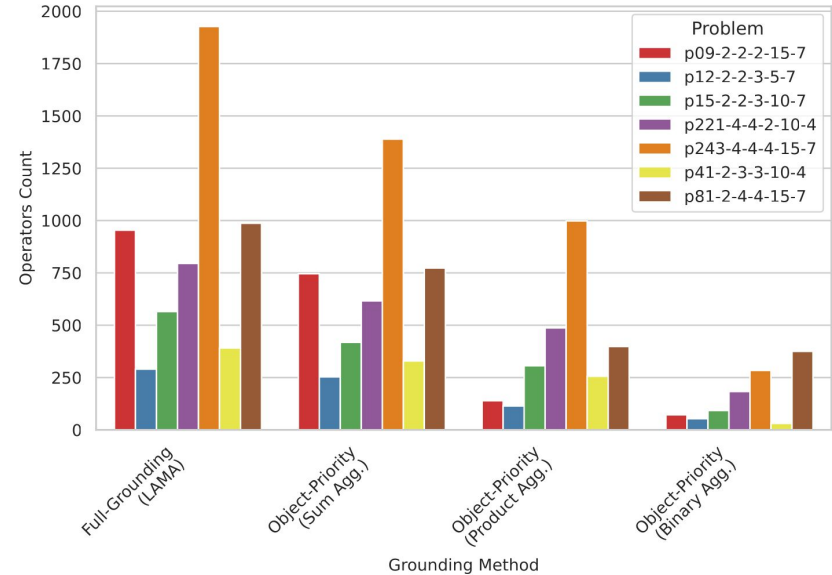
RR queue with stopping criteria plus 10% incrementing actions

Empirical Results

Grounded Actions: Blocksworld Domain



Grounded Actions: Satellite Domain



grounded task size and solvable



of operators (Blocksworld & Satellite)

Object Priorities: Different Approaches

Sub-symbolic Learning

WL Embeddings Chen et al., (2024)

GNNs Ståhlberg, Bonet, and Geffner (2022, 2023)

LLMs to emit priority function (Correa, Pereira, and Seipp 2025)

Symbolic Rules:

Lifted Landmark Heuristics (e.g., LM-cut)

Lifted Delete Relaxation Heuristics k (Wichlacz et al. 2023)

Unary relaxation by Lauer et al., (2021)

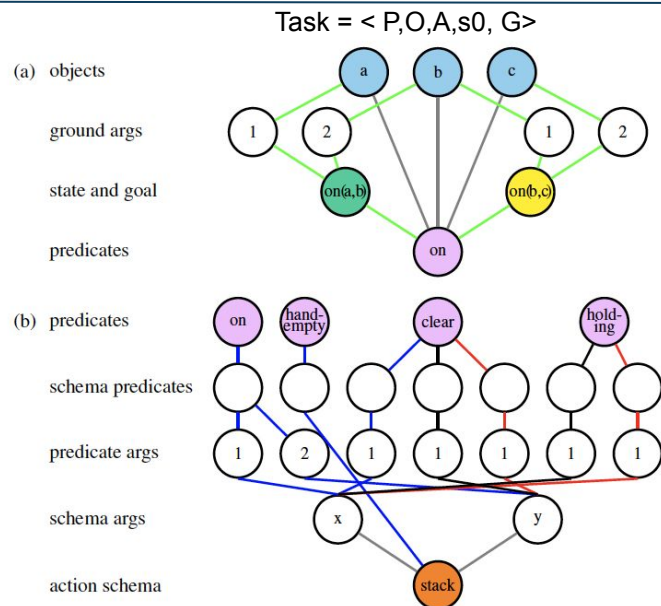


Figure 3: LLG instance subgraph (a) and schema subgraph (b) with graph layer descriptions of a Blocksworld instance. Chen et al., (2024)

Conclusion

- ❑ **Partial Grounding** reduces total task size while preserving solvable tasks.
- ❑ **Per-parameter efficiency**: decomposes priority scores parameter by parameter avoids the combinatorial bottleneck
- ❑ **Sound by construction**: provides a safe fallback recovery path back to Full Grounding
- ❑ **Work in Progress**: Different approaches to compute **object priorities**
 - ❑ **Lifted LM cut Heuristics**
 - ❑ **WL and GNN**

Thank you!

Questions & Discussion

Mariam Musavi

mariam.musavi@liu.se

Johannes K. Fichte

johannes.fichte@liu.se

Daniel Gnad

daniel.gnad@uni-heidelberg.de