# GOFAI

**Álvaro Torralba**[1], **Daniel Gnad**[2]
[1] Aalborg University, Denmark
[2] Linköping University, Sweden
alto@cs.aau.dk, daniel.gnad@liu.se

## Abstract

The main idea behind the GOFAI planner is to use *partial grounding* to avoid building the full SAS+ planning task before looking for a plan. GOFAI is implemented in Fast Downward (FD) and uses its search component without changes. The key component of GOFAI is FD's translator, which instantiates the given PDDL planning task into a fully grounded SAS+ representation. We adapt this part by learning, for a specific domain, which actions of a task are most relevant to find a plan. With this, we focus the grounding process onto these actions and avoid grounding actions that are predicted to be less relevant. We employ several different approaches to learn action-relevance predictors and pick the best such model using the algorithm configuration tool SMAC. We also let SMAC select the search configuration of FD from a predefined set that best interacts with the partial grounding model.

## Planner Description

Most state-of-the-art classical planners translate a given PDDL planning task to a fully grounded representation before even starting to look for a plan (Helmert 2006; Ramirez, Lipovetzky, and Muise 2015; Froleyks, Balyo, and Schreiber 2019). In GOFAI we build on our initial work on *partial grounding* that avoids constructing the full SAS+ representation of the PDDL input task (Gnad et al. 2019).

From Gnad et al. (2019) we adopt the learning of models based on relation rule features using off-the-shelf machine learning (ML) techniques such as linear regression or support vector machines from the SciKit Learn library[1]. We also adopt the learning of relational tree using the Aleph tool[2]. With both approaches, we learn to predict the relevance of actions that are about to be grounded. We rank these actions according to their relevance, as shown in Algorithm 1, which instantiates the priority queue in line 11. The partial grounding terminates if (at least) the goal is reachable under delete-relaxation semantics (line 4: $G \subseteq F$) and some stopping condition triggers, or when all actions have been grounded.

As stopping conditions, we only use fairly simple criteria based on the number of actions grounded when the goal becomes relaxed reachable. We then either terminate immediately, or ground an additional $x\%$ of actions.

[1]See https://scikit-learn.org/.

[2]See https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html.

**Algorithm 1:** Partial grounding algorithm.

```
   Input: A lifted task $\Pi^{PDDL} = (\mathcal{P}, \mathcal{A}, \Sigma^C, \Sigma^O, I, G)$
   Output: A STRIPS task $\Pi = (F, O, I, G)$
1  $q \leftarrow$ I ;
2  $F \leftarrow \emptyset$ ;                    // Processed facts
3  $O \leftarrow \emptyset$ ;                // Processed operators
4  while $\neg(q$.empty() $\vee G \subseteq F) \wedge \neg$Stop do
5      if $q$.containsFact() then
6          $f \leftarrow q$.popFact() ;
7          $F \leftarrow F \cup \{f\}$ ;
8          for $o \notin O \wedge \mathsf{pre}(o) \subseteq F$ do
9              $q$.insert($o$) ;
10     else
11         $o \leftarrow q$.popHighRelevanceOperator() ;
12         $O \leftarrow O \cup \{o\}$ ;
13         for $f \notin F \wedge f \in \mathsf{add}(o)$ do
14             $q$.insert($f$) ;
15 return $(F, O, I, G)$
```

In addition to the models that predict the action relevance, we also try to learn special rules that identify actions that should *always*, respectively *never*, be grounded. The latter actions will be completely discarded. Actions that should always be grounded get a maximum relevance score and additionally trigger the stopping condition: as long as there are still such actions in the queue, the grounding continues. We refer to these rules as *hard rules*, as they do not only give a relevance score to actions. The hard rules are obtained using Aleph by querying for expressions that exactly separate useful from non-useful actions on a set of training instances.

The overall training of models is wrapped into the algorithm configuration tool SMAC (Hutter, Hoos, and Leyton-Brown 2011). We train all from a predefined set of models in the same way as in done in Gnad et al. (2019), and then let SMAC decide on which kind of priority queue is used (see Gnad et al. (2019) for details), and which stopping condition to use. Furthermore, SMAC optimizes the set of hard rules that should be used during grounding, as these rules are obtained from a subset of the training instances, so might not generalize. The SMAC optimization uses a disjoint set of instances, so can possible identify issues.

Moreover, we provide a list of search configurations to

SMAC and let it chose the best one for a given domain and partial-grounding model. This list consists of the LAMA-first configuration (Richter and Westphal 2010) and all configurations that are part of the Fast Downward Stone Soup portfolio from IPC 2018 (Seipp and Röger 2018).

We extend the standard preprocessing of Fast Downward with the $h^2$-based task simplification by Alcázar and Torralba (2015), which removes irrelevant and unreachable facts and actions from the task before invoking the search.

Partial grounding is an incomplete approach to planning in general. Hence, we adopt the incremental grounding scheme of Gnad et al. (2019), which grounds more actions in later iterations if the search fails to find a plan. For every search iteration, we limit the time to 5min. If the search fails, the next round of partial-grounding instantiates $10,000$ additional actions. The outcome of the SMAC optimization can be a set of planner configurations. We equally distribute 25min across these configurations and let a standard LAMA configuration (with full grounding) run for the last 5min.

**Multi-Core Track** In the multi-core track, we parallelize the generation of training data, i.e., solving the training instances, as well as the SMAC optimization. The planning phase still only uses a single core.

## References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Froleyks, N. C.; Balyo, T.; and Schreiber, D. 2019. PASAR - Planning as Satisfiability with Abstraction Refinement. In Surynek, P.; and Yeoh, W., eds., *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, 70–78. AAAI Press.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7602–7609. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In Coello, C. A. C., ed., *Proceedings of the Fifth Conference on Learning and Intelligent OptimizatioN (LION 2011)*, 507–523. Springer.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Seipp, J.; and Röger, G. 2018. Fast Downward Stone Soup 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 80–82.