# Towards Logical Model-based Reinforcement Learning: Lifted Operator Models

**Corentin Sautier**[1,2] , **Don Joven Agravante**[1] , **Michiaki Tatsubori**[1]
[1]IBM Research
[2]MINES ParisTech - PSL Research University, Paris, France

## Abstract

Model-based reinforcement learning has produced significant state-of-the-art results in recent years. However, current models are still opaque and difficult to integrate with external knowledge bases. To address these issues, we envision a two-stage process where deep learning first transforms raw observations into a logical state. Having this logical representation provides interpretability and an insertion point for external knowledge. As a second stage, we propose to leverage concepts from classical planning within the model-based reinforcement learning framework. In particular, we adapt lifted operators as our model to be learned and then plan over this model with PDDL planners. This paper focuses on the second stage where we show how we can learn the lifted operator models from an imperfect semantic parser.

## 1 Introduction

Reinforcement Learning (RL) is the machine learning paradigm whereby an agent learns the best way to interact with an environment. This is done by taking actions and observing the results. The integration of deep learning with RL has shown to be very effective, in particular when interacting with complex environments such as directly from the stream of images in video games [Mnih *et al.*, 2013]. The initial popularity of so-called deep RL was due to model-free RL, where the deep neural networks are used in an *end-to-end* manner. Although this method is very effective, it came along with several well-known issues such as the lack of explainability and need for huge amounts of interaction data [Dulac-Arnold *et al.*, 2019].

To address the shortcomings of deep RL, research has recently shifted the focus to using deep learning with model-based RL. The philosophy of the approach is to first learn a model of the environment dynamics and then to plan over this model. Recent results have shown that this is an even more promising approach [Schrittwieser *et al.*, 2019; Łukasz Kaiser *et al.*, 2020]. In particular, these results have shown significant improvements in data efficiency. However, the models are still fairly opaque making it difficult to interpret. To improve this, we propose to incorporate concepts from classical planning - in particular symbolic logic as an intermediate representation.

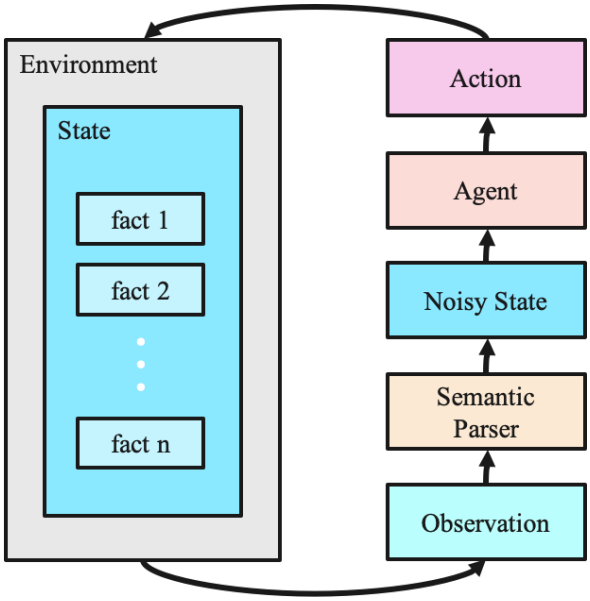


Figure 1: Model-based architecture with logical states

Figure 1 shows our envisioned framework and the problem setting. The left side depicts that the environment state can be sufficiently approximated as a set of logical facts. Continuing in the bottom right, the agent can get observations of the environment (e.g. images). Here, we assume that we have a *semantic parser* that converts these observations into a logical form. We assume that the semantic parsing is good, but won't be perfect, hence we only have a noisy representation of the real state. The task of our agent is to learn from the noisy states and produce good actions in the environment. Our framework is a two-stage process - a semantic parser and an agent that learns from noisy logical states. This paper concentrates on the agent but semantic parsing is a growing and rapidly improving research area in both text [Ge and Mooney, 2009] and images [Herzig *et al.*, 2018]. We anticipate that bigger pre-trained models will soon open-up several application areas, one of which is our problem setting.

Since our agent takes as input noisy logical states based

on interacting with an environment, we are at the intersection of model-based RL and classical planning. In this paper, our approach is to formulate the learned model as lifted operators of classical planning. This would allow us to use planners that can provide guarantees about consistency, safety and optimality. In the classical planning literature, operator learning has been shown in the context of using partial or impractical data, such as partial states or sequences of action only as it would limit the amount of human work in specifying the problem [Yang *et al.*, 2007; Cresswell and Gregory, 2011; Cresswell *et al.*, 2013]. Similarly, data-agnostic methods have been developed, often presented as a satisfiability problem [Aineto *et al.*, 2018; Zhuo *et al.*, 2019].

In contrast to our outlined approach, there are two notable works that exist in the literature that consists of training a model while still possibly benefiting from the structure of symbolic logic. First, [Evans and Grefenstette, 2017] combines Inductive Logic Programming with Neural Networks to enable this. Second, [Asai and Fukunaga, 2017] tries to solve a similar problems to ours, end-to-end, using an automatically learned representation, and a classical planner. However, the ability to solve such a problem end-to-end seem to again come at the price of explainability from a human perspective.

In this paper, we discuss solutions to convert a few noisy high-level environment states into operators' knowledge for planning problems, and use them to solve the Tower of Hanoi provided by PDDLGym [Silver and Chitnis, 2020]. We show that lifting propositions can help an effective learning of the operators, especially when considering noisy data. We discuss methods to learn operators, with Neural Network, and present a few use-cases where Neural Networks can bring a real benefit in dealing with operators with noisy data. We tackle the preconditions learning in a novel way, using Feature Importance, and prove it has the ability to disambiguate preconditions from other frequent propositions which the most usual methods fail to do.

## 2 Problem setting and formulation

We consider a deterministic environment $E$, that, even if only observable through images, is based on an internal logic state, $\tilde{s} \in S$, or can be approximated as such. Here, we define our logical states to consist of predicate functions grounded on some objects. For example, *on(disc1, peg1)* is a proposition composed of the predicate *on*, grounded on the objects *disc1* and *peg1*. Our full logical state is defined as a conjunction of every proposition's boolean value at a given time.

We keep the basic RL problem setting wherein the agent interacts with environment, $E$, by deciding actions, $a \in A$, to take based on observations, $o \in O$. The environment transitions the state based on the action taken such that the dynamics of the next state is $\tilde{s}' = T(\tilde{s}, a)$. Two particularities of our problem setting are now added onto this base setting. First, we assume that we have a good but imperfect semantic parser, $\phi$, that produces approximates of the state, $s$, from $o$, that is $s = \phi(o)$. Second, the environment dynamics, $T$, can be re-formulated as planning *operators* with *preconditions* and *effects*. Preconditions are the set of logical statements needed for an action to be valid. The effects are the set of logical statements that change between $\tilde{s}'$ and $\tilde{s}$. We show these in subsection 2.1

We follow the model-based RL setting where we learn the model (i.e. the operators) from a dataset of triplets, $(s, a, s') \in S \times A \times S$. Note that we can only collect the approximates of state, $s$, instead of the actual internal state $\tilde{s}$. A triplet is *valid* if $\tilde{s}$ respects the preconditions of $a$, and from the application of the effects it comes $\tilde{s} \neq \tilde{s}'$. By extension, we can also define *valid actions* relative to the triplet validity. In the problem of the Tower of Hanoi for instance, trying to place a large disc on a smaller one would be an invalid action.

Once we have learned the operators from the triplets, our agent has to plan over this model to produce *good* actions. Since we follow here the PDDL planning formulation, we only need the initial state $s_i \in S$ and a goal state $s_g \in S$ together with the operators. We assume that the initial state we obtain is sufficiently good. As for the goal state, although it is possible to relate this to the RL reward function, we leave this as a future work and assume here that the goal state is specified beforehand.

### 2.1 Operators

An action, similarly to a proposition, is composed of an operator predicate, grounded on objects. An operator is an abstract representation of every possible action, and contains the information about the preconditions and the effects. The PDDLgym implementation of the Tower of Hanoi [Silver and Chitnis, 2020], only has a single operator: **move**, of arity 3, defined in PDDL formatting, by:

:**action** *move*
:**parameters** (?disc ?from ?to)

| :**precondition** | :**effect** |
|---|---|
| (and | (and |
| (*smaller* ?to ?disc) | (*clear* ?from) |
| (*on* ?disc ?from) | (*on* ?disc ?to) |
| (*clear* ?disc) | (not (*on* ?disc ?from)) |
| (*clear* ?to)) | (not (*clear* ?to)))) |

The action *move* defines 3 *parameters*, whose state must match a list of propositions in the *preconditions* to consider the action valid, and will be affected by the *effects* in the state following the action. Operators can fully define all state transitions and the main task in this paper is to learn an equivalent representation of this operator.

### 2.2 Lifting of an action

The PDDL domain defines operators by using *parameters*, which, once grounded on actual objects becomes an action. The intuition is that this operator representation contains general *model* knowledge, whereas actions would only inform about the current state, making the latter less efficient.

Practically, every objects the action is grounded on can be converted into an abstract variable, specifying only their index in the action and propositions related to other objects should be discarded, reducing their number (Table 1). Two different propositions can appear similar once object-specific details are abstracted (Table 2) thus improving both data efficiency and consistency to unseen states.

Table 1: Lifting process

| Grounded | Lifted |
|---|---|
| :**action** | :**action** |
| move(disc1, disc2, peg3) | move(?v1, ?v2, ?v3) |
| :**state** | :**state** |
| clear(disc1) | clear(?v1) |
| clear(peg2) | |
| clear(peg3) | clear(?v3) |
| on(disc1, disc2) | on(?v1, ?v2) |
| on(disc2, disc3) | |
| smaller(disc1, disc2) | smaller(?v1, ?v2) |
| smaller(disc1, peg3) | smaller(?v1, ?v3) |
| smaller(disc2, peg3) | smaller(?v2, ?v3) |
| ... | |

Table 2: Generalization capabilities of the lifting

| Grounded | Lifted |
|---|---|
| :**action 1** | :**action 1** |
| move(disc1, disc2, peg3) | move(?v1, ?v2, ?v3) |
| :**state 1** | :**state 1** |
| on(disc1, disc2) | on(?v1, ?v2) |
| :**action 2** | :**action 2** |
| move(disc1, disc3, peg3) | move(?v1, ?v2, ?v3) |
| :**state 2** | :**state 2** |
| on(disc1, disc3) | on(?v1, ?v2) |

# 3 Learning of the operators

We split this section into the operator preconditions and effects. However, we first discuss the effects since it is simpler.

## 3.1 Operator effects

Let's consider a valid triplet $(s, a, s')$. The effects of $a$ are all the proposition from $s$ that have become true or false in $s'$. It comes that the effects of $a$ to $s$ can be found by taking $s' - s$. If we apply lifting on this difference with perfect states, we obtain the exact correct operator's effects, meaning that, in the absence of noise only a single valid action is required to *learn* them.

In the presence of noise, a simple strategy would be to average those effects found from multiple triplets. This can already give a strong statistical baseline. Furthermore, we can also use them as noisy labels to train a neural network. Such a neural network, would need to be inputted the operator itself, which we convert in a one-hot encoding, and we also input the boolean value of every possible propositions for the lifted state. It might seem unnecessary, but we believe inputting the relevant part of a state can help the network recover from the noise. It will also serves the purpose of allowing to predict the preconditions.

## 3.2 Operator preconditions

Let's consider a valid triplet $(s, a, s')$ and assume first the state are perfectly correct. The preconditions of $a$ are the boolean value of the proposition from $s$ that were needed for it to be valid. The task is to find which of the propositions in $s$ are necessary, and which are just coincidental. To do this, let's consider an operator $\omega$, if we name $P$ the set of all possible propositions' boolean value, and $pr$ the set of all preconditions of $\omega$, and $V_\omega$ the subset of valid triplets whose actions are grounded from $\omega$, by the definition of the precondition, it comes:

$$\forall p \in P,\, p \in pr \implies \forall (s, a, s') \in V_\omega,\, p \in s,$$

and by contraposition:

$$\forall p \in P,\, \exists (s, a, s') \in V_\omega \mid p \notin s \implies p \notin pr.$$

Essentially, that means if we can find a valid triplet in which state a proposition is not, this proposition is not a precondition of the action. It comes that a precondition is the intersection of the lifted propositions in all the valid triplets' state of the dataset. Given a diverse enough dataset, this allows the finding of working operators' preconditions.

When we consider noisy states, a triplet's state might be missing a proposition that is actually a precondition, if the noise removed it. Thus instead of considering the preconditions are the propositions present in *every* valid triplet's states, we consider those present in *most* valid triplets' states, defining a cutoff proportion $p_{cutoff}$, a precondition is a proposition's boolean value seen in more than $p_{cutoff}$ of the triplets in the dataset.

Intuitively, there can be a benefit in using invalid triplets as well, as it could be used to remove spurious propositions that are actually decorrelated to the validity of an action. We built and trained a binary classifier, predicting an action validity from the state and operator input.

Obtaining the preconditions can be trimmed down to finding which propositions – or neural network *features* – are responsible for the discrimination between a valid and an invalid triplet. We used a variation of the Feature Importance method (Algorithm 1) [Breiman, 2001] to attribute an importance score to each proposition, and consider a precondition for an operator every proposition above a certain threshold.

**Algorithm 1:** Feature Importance

```
importance = dict()
for (s, a, s') in dataset do
    reference = model_validity(s, a)
    if reference > 0.5 then
        for p in P do
            if p in s then
                prediction = model_validity(s - p, a)
            else
                prediction = model_validity(s + p, a)
            end
            importance[a.operator] += d(prediction,
              reference)
        end
    end
end
```

*model_validity* is the validity prediction by the model
*d* is a distance.
*a.operator* is the operator of the action a

# 4 Results

## 4.1 Grounded or Lifted Representations

We trained a lifted and a grounded model on 2000 triplets, roughly 10% of which are valid, with uniform noises. The uniform noise present the probability that each proposition's boolean state is inversed. The task trained on was a state transition, meaning by inputting a state and an action, we tried to output the next state.

We report the F1 score between the prediction of the next state, and the ground truth for 1000 valid triplets.

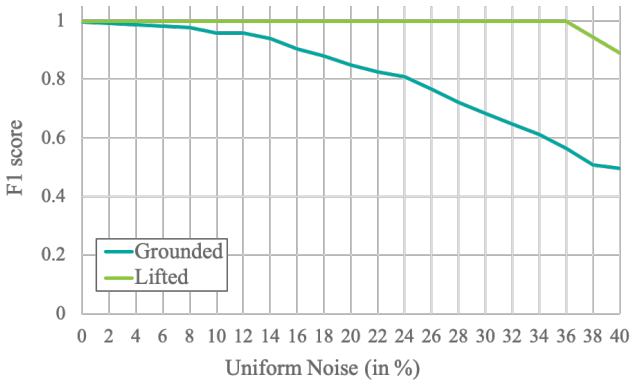


Figure 2: Evolution of Grounded and Lifted F1 score with noise

The results (Figure 2) show that, in absence of noise, both models have similar capabilities, however, the grounded model will fit to the noise, making it very little resilient.

## 4.2 Results for the effects

The baseline method will consider as effects the propositions seen in a majority of the valid examples' effects. Since those effects are the difference between a state after and before an action, they are affected by noise on both state.

We report the solving rate of the planner, using effects learned from 4000 noisy triplets and ground-truth preconditions to complete the domain knowledge.

Table 3: Results for the solving rate while predicting operators' effect with uniform noise

| Noise | 0.25 | 0.26 | 0.27 | 0.28 | 0.29 | 0.30 |
|---|---|---|---|---|---|---|
| Baseline | 95% | 75% | 60% | 25% | 10% | 0% |
| Our Method | 70% | 75% | 55% | 60% | 60% | 50% |

Our method seems able to learn on labels more often wrong than right. In our understanding, there are two phenomenon that could explain that:

- The effects being a logic form, its propositions are not completely decorrelated from each other. It is possible the network learn that some propositions have to be put together.
- The ML method has in its inputs the previous state, which might help the network to learn some noise control in the very noisy effects, from the less noised previous state.

A second point to notice is the inherent instability of the Neural Network, with noisy labels, as the solving rate is very variable even with lower noise amounts, that statistical method can solve quite reliably.

## 4.3 Results for the preconditions

The non-Deep Learning solution for the preconditions has a major flaw. There can be situations were no value of $p_{cutoff}$ could solve the problem. If a proposition $p$ is very frequent, but not a precondition of an operator $\omega$, it could still be seen in more than $p_{cutoff}$ of the valid triplets with $\omega$ and thus be considered by this method as a precondition. Increasing the cutoff proportion can fix this problem, but at the same time reduces noise resilience to false negative.

In order to test this inability of the non-Deep Learning solution, we need to have an example of a proposition, not belonging to the preconditions of an operator, but still very frequent. By default, in the PDDLGym [Silver and Chitnis, 2020] implementation of the Tower of Hanoi, there is no such case, we thus design a noise specifically to simulate those conditions. In this noise, we set to true randomly, with a probability of 0.6 all possible *smaller* propositions, and don't modify the others. This makes that in a lifted valid example of the action, some *smaller* propositions will appear very often, regardless of their precondition status. We used a cutoff value for the intersection method of 0.75, having found it to be an effective value for various amounts of evenly distributed noise. We used the same 4000 training examples for both the intersection and the ML methods to learn the preconditions, and completed the domain knowledge with ground-truth effects. We report here the solving rate on 20 games.

The results are that our methods could solve **85%** of the games, while the non-Deep Learning one had a solving rate of only **10%**

The binary classifier's success is explainable by these propositions being decorrelated with the validity of an action, even when very frequent, and this translates in having a low feature importance score. Fundamentally, the intersection method fails here because it does not use invalid actions to gather knowledge.

# 5 Conclusion

Operators knowledge can be obtained from few formatted states, even with a significant amount of noise. Lifting the state observations can benefit greatly the models, in terms of generalization, and noise resilience. The learning of effects benefit from exact labels that are the difference between two consecutive states. As for the preconditions, no fail-proof solution exist, we presented two different methods, a first very data-efficient, but lacking the ability to solve all problems, and a second, theoretically more universal, but relying on some unstable techniques – Deep Neural Networks with noisy labels and Feature Importance.

Finally, we proved empirically that Neural Networks can bring a real benefit to operators learning, both in terms of theoretical capabilities, and noise resilience, and proved those findings empirically on the problem of the Tower of Hanoi.

## References


[Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and E. Onaindia. Learning strips action models with classical planning. *Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 2018.

[Asai and Fukunaga, 2017] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *Thirty-Second AAAI Conference on Artificial Intelligence*, 04 2017.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Cresswell and Gregory, 2011] Stephen Cresswell and Pete Gregory. Generalised domain model acquisition from action traces. *Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 6 2011.

[Cresswell *et al.*, 2013] Stephen Cresswell, T. Mccluskey, and Margaret West. Acquiring planning domain models using locm. *The Knowledge Engineering Review*, 28, 06 2013.

[Dulac-Arnold *et al.*, 2019] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of Real-World Reinforcement Learning. *arXiv*, 2019.

[Evans and Grefenstette, 2017] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61, 11 2017.

[Ge and Mooney, 2009] Ruifang Ge and Raymond Mooney. Learning a compositional semantic parser using an existing syntactic parser. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 611–619, 08 2009.

[Herzig *et al.*, 2018] Roei Herzig, Moshiko Raboh, Gal Chechik, Jonathan Berant, and A. Globerson. Mapping images to scene graphs with permutation-invariant structured prediction. *ArXiv*, abs/1802.05451, 2018.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[Schrittwieser *et al.*, 2019] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model, 2019.

[Silver and Chitnis, 2020] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems, 2020.

[Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2):107 – 143, 2007.

[Zhuo *et al.*, 2019] Hankui Zhuo, Jing Peng, and Subbarao Kambhampati. Learning action models from disordered and noisy plan traces. *ArXiv*, abs/1908.09800, 2019.

[Łukasz Kaiser *et al.*, 2020] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.