Appendix

1 Differences between DRLM and ∂ILP

There are 2 major differences between DRLM and ∂ILP. First, each deduction step of ∂ILP outputs the probabilistic sum of the valuation generated in this step and the valuation in last step, which makes definitions that require less steps of deduction repeatedly affect final outputs, namely, the result valuation. In RL tasks, this give the agent strong incentive to use simpler strategies with less logic chaining and therefore easier be trapped in local optima. We instead make the agent treat valuation produced by last step deduction only as input. The output of each step is simply the sum of deduced valuation and the initial inputs $\boldsymbol{e}_0$ that encloses both the state and background information in NLRL. Notably, the valuations deduced in last step will not be forgot because they will be derived again in current step.

Another difference is about the weights, in ∂ILP, the weights are assigned to combinations of clauses for each intensional predicate, whereas DRLM assign them to individual clauses. It is easy to see that later approach will use much less variables. Suppose there are n predicates each defined by m clauses, and each clause has r possible definitions. The ∂ILP implementation will use nr^m variables while the DRLM will only consume nmr of them. ∂ILP used the memory expensive one because they use elementary max to combine valuations of different clauses defining the same predicate, in which case they found conclusions of the two clauses will over-write each other when they are combined and thus the gradient flow is truncated. To prevent truncated gradient flow, they thus choose to assign variables of combinations of clauses directly. Such choice makes ∂ILP consuming huge amount of memory while at the same time constraint the program templates to define two and only 2 clauses for each intensional predicate. We use the probabilistic sum in place of max to prevent the truncated gradient flow problem while keeping the advantage of smaller, memory cheaper and more flexible model.

2 policy interpretation of other tasks

UNSTACK induced policy: The policy induced by NLRL in *UNSTACK* task is:

$$\begin{aligned} &0.972 : move(X,Y) \leftarrow isFloor(Y), pred(X) \\ &0.987 : pred(X) \leftarrow pred2(X), top(X) \\ &0.997 : pred2(X) \leftarrow on(X,Y), on(Y,Z) \end{aligned} \tag{1}$$

We only show the invented predicates that are used by the action predicate and the definition clause with high confidence (larger than 0.3) here. The $pred2(X)$ means the block X is on top of another block (the block is not directly on the floor). The $pred(X)$ means the block X is in the top position of a column of blocks and it is not directly on the floor, which basically indicates the block to be moved. The action predicate $move(X,Y)$ simply move the top block in any column with more than 1 block to the floor.

ON induced policy: The induced policy of the *ON* task is:

$$\begin{aligned} &1.000 : move(X,Y) \leftarrow top(X), pred(X,Y) \\ &1.000 : move(X,Y) \leftarrow top(X), goalOn(X,Y) \\ &0.947 : pred(X,Y) \leftarrow isFloor(Y), pred2(X) \\ &1.000 : pred2(X) \leftarrow on(X,Y), on(Y,Z) \end{aligned} \tag{2}$$

The goal of *ON* is to move block a onto b, while in the training environment the block a is at the bottom of the whole column of blocks. The strategy NLRL agent learned is to first unstack all the blocks and then move a onto b. The first clause of move $move(X,Y) \leftarrow top(X), pred(X,Y)$ implements the unstack procedures, where the logics are similar to the *UNSTACK* task. The second clause $move(X,Y) \leftarrow top(X), goalOn(X,Y)$ tells if the block X is already movable (there is no blocks above), just move X on Y. This strategy can deal with most of the circumstances and is optimal in the training environment. Whereas, we can also construct non-optimal case where unstacking all the blocks are not necessary or if the block b is below the block a, e.g., $((b,c,a,d))$.

Windy cliff-walking induced policy: The induced policy of the windy cliff-walking task is:

$$\begin{aligned} &0.999 : down() \leftarrow current(X,Y), last(X) \\ &0.472 : right() \leftarrow current(X,Y), succ(Z,Y) \\ &0.628 : right() \leftarrow current(X,Y), succ(Z,X) \\ &0.966 : up() \leftarrow current(X,Y), zero(X) \end{aligned} \tag{3}$$

Differing from the original cliff-walking, the agent tends be more conservative. It tends to move upwards when it is on the left edge of the field. By doing so, there is less risk of falling to the cliff when it approaching the right edge.