Supplemental Materials

A Details on Neurally-Guided Symbolic Abstraction

We here provide details on the neurally-guided symbolic abstraction algorithm.

A.1 Algorithm of Neurally-Guided Symbolic Abstraction

We show the algorithm of neurally-guided symbolic abstraction in Algorithm 1.

Algorithm 1 *Neurally-Guided Symbolic Abstraction*

Input: $\mathcal{C}_0, \pi_\theta$, hyperparameters (N_{beam}, T_{beam})
1: $\mathcal{C}_{to\_open} \leftarrow \mathcal{C}_0$
2: $\mathcal{C} \leftarrow \emptyset$
3: $t = 0$
4: **while** $t < T_{beam}$ **do**
5: $\quad \mathcal{C}_{beam} \leftarrow \emptyset$
6: $\quad$ **for** $C_i \in \mathcal{C}_{to\_open}$ **do**
7: $\quad\quad \mathcal{C} = \mathcal{C} \cup \{C_i\}$
8: $\quad\quad$ **for** $R \in \rho(C_i)$ **do**
$\quad\quad\quad$ `# Evaluate each clause`
9: $\quad\quad\quad score = eval(R, \pi_\theta)$
$\quad\quad\quad$ `# select top-k rules`
10: $\quad\quad\quad \mathcal{C}_{beam} = top\_k(\mathcal{C}_{beam}, R, score, N_{beam})$
$\quad$ `# selected rules are refined next`
11: $\quad \mathcal{C}_{to\_open} = \mathcal{C}_{beam}$
12: $\quad t = t + 1$
return $\mathcal{C}$

A.2 Rule Generation

At line 8 in Algorithm 1, given action rule C, we generate new action rules using the following refinement operation:

$$\rho(C) = \{X_A \leftarrow X_S^{(1)}, \dots X_S^{(n)}, Y_S \mid Y_S \in \mathcal{G}_S^* \ \wedge \ Y_S \neq X_S^{(i)}\}, \tag{7}$$

where $\mathcal{G}_S^*$ is a non-ground state atoms. This operation is a specification of *(downward) refinement operator*, which a fundamental technique for rule learning in ILP [Nienhuys-Cheng and de Wolf, 1997], for action rules to solve RL tasks.

We use mode declarations [Muggleton, 1995, Cropper et al., 2022] to define the search space, *i.e.* $\mathcal{G}_S^*$ in Eq. 7 which are defined as follows. A mode declaration is either a head declaration $\mathtt{modeh}(\mathtt{r}, \mathtt{p}(\mathtt{mdt_1}, \dots, \mathtt{mdt_n}))$ or a body declaration $\mathtt{modeb}(\mathtt{r}, \mathtt{p}(\mathtt{mdt_1}, \dots, \mathtt{mdt_n}))$, where $\mathtt{r} \in \mathbb{N}$ is an integer, $\mathtt{p}$ is a predicate, and $\mathtt{mdt_i}$ is a mode datatype. A mode datatype is a tuple $(\mathtt{pm}, \mathtt{dt})$, where $\mathtt{pm}$ is a place-marker and $\mathtt{dt}$ is a datatype. A place-marker is either $\#$, which represents constants, or $+$ (resp. $-$), which represents input (resp. output) variables. $\mathtt{r}$ represents the number of the usages of the predicate to compose a solution. Given a set of mode declarations, we can determine a finite set of rules to be generated by the rule refinement.

Now we describe mode declarations we used in our experiments. For Getout, we used the following mode declarations:

$$\mathtt{modeb}(2, \mathtt{type}(-\mathtt{object}, +\mathtt{type}))$$
$$\mathtt{modeb}(1, \mathtt{closeby}(+\mathtt{object}, +\mathtt{object}))$$
$$\mathtt{modeb}(1, \mathtt{on\_left}(+\mathtt{object}, +\mathtt{object}))$$
$$\mathtt{modeb}(1, \mathtt{on\_right}(+\mathtt{object}, +\mathtt{object}))$$
$$\mathtt{modeb}(1, \mathtt{have\_key}(+\mathtt{object}))$$
$$\mathtt{modeb}(1, \mathtt{not\_have\_key}(+\mathtt{object}))$$

For 3Fishes, we used the following mode declarations:

$$
\begin{aligned}
&\mathtt{modeb}(2, \mathtt{type}(-\mathtt{object}, +\mathtt{type}))\\
&\mathtt{modeb}(1, \mathtt{closeby}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_top}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{at\_bottom}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_left}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_right}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{bigger\_than}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{high\_level}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{low\_level}(+\mathtt{object}, +\mathtt{object}))
\end{aligned}
$$

For Loot, we used the following mode declarations:

$$
\begin{aligned}
&\mathtt{modeb}(2, \mathtt{type}(-\mathtt{object}, +\mathtt{type}))\\
&\mathtt{modeb}(2, \mathtt{color}(+\mathtt{object}, \#\mathtt{color}))\\
&\mathtt{modeb}(1, \mathtt{closeby}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_top}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{at\_bottom}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_left}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{on\_right}(+\mathtt{object}, +\mathtt{object}))\\
&\mathtt{modeb}(1, \mathtt{have\_key}(+\mathtt{object}))
\end{aligned}
$$

B Additional Results

B.1 Weights learning

Fig. 6 shows the NUDGE agent $\pi_{(\mathcal{C},\mathbf{W})}$ parameterized by rules $\mathcal{C}$ and weights $\mathbf{W}$ before training (top) and after training (bottom) on the GetOut environment. Each element on the x-axis of the plots corresponds to an action rule. In this examples, we have 10 action rules $\mathcal{C} = \{C_0, C_1, \ldots, C_9\}$, and we assign $M = 5$ weights *i.e.* $\mathbf{W} = [\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_4]$. The distributions of rule weighs with *softmax* are getting peaked by learning to maximize the return. The right 4 rules are redundant rules, and theses rules get low weights after learning.

B.2 Deduction Pipeline

Fig. 7 provides the deduction pipeline of a NUDGE agent on 3 different states. Facts can be deduced from an object detection method, or directly given by the object centric environment. For state #1, the agent chooses to jump as the `jump` action is prioritized over the other ones and all atoms that compose this rules' body have high valuation (including `closeby`). In state #2, the agent chose to go **left** as the rule `left_key` is selected. In state #3, the agent selects **right** as the rule `right_door` has the highest forward chaining evaluation.

B.3 Policies of every logic environment.

We show the logic policies obtained by NUDGE in GetOut, 3Fishes, and Loot in Fig. 8, *e.g.* the first line of GetOut, "$\mathtt{0.574}:\mathtt{jump(X)}\text{:-}\mathtt{closeby(O1,O2)},\mathtt{type(O1,agent)},\mathtt{type(O2,enemy)}.$", represents that the action rule is chosen by the weight vector $\mathbf{w}_1$ with a value 0.574. NUDGE agents have several weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$ and thus several chosen action rules are shown for each environment.

Figure 6: Weights on action rules via softmax before training (top) and after training (bottom) on NUDGE in GetOut. Each element on the x-axis of the plots corresponds to an action rule. NUDGE learns to get high returns while identifying useful action rules to solve the RL task. The right 5 rules are redundant rules, and theses rules get low weights after learning.

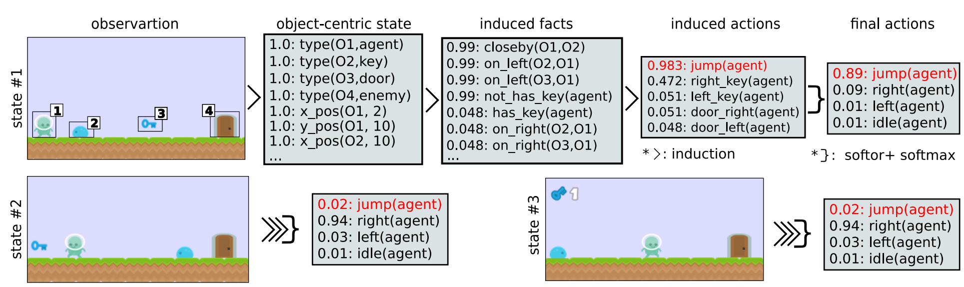

Figure 7: The logic reasoning of NUDGE agents makes them interpretable. The detailed logic pipeline for the input state #1 of the *Getout* environment and the condensed action selection for state #2 and state #3.

C Illustrations of our environments

We showcase in Fig. 9 one state of the 3 object-centric environments and their variations. In **GetOut** (blue humanoid agent), the goal is to obtain a key, then go to a door, while avoiding a moving enemy. **GetOut-2En** is a variation with 2 enemies. In **3Fishes**, the agent controls a green fish and is confronted with 2 other fishes, one smaller (that the agent need to "eat", *i.e.* go to) and one bigger, that the agent needs to dodge. A variation is **3Fishes-C**, where the agent can eat green fishes and dodge red ones, all fishes have the same size. Finally, in **Loot**, the (orange) agent is exposed with 1 or 2 chests and their corresponding (*i.e.* same color) keys. In **Loot-C**, the chests have different colors. All 3 environment are *stationary* in the sense of Delfosse et al. [2021].

```
# GetOut
0.574:jump(X):-closeby(O1,O2),type(O1,agent),type(O2,enemy).
0.315:right_go_to_door(X):-have_key(X),on_left(O1,O2),type(O1,agent),type(O2,door).
0.296:right_go_to_door(X):-have_key(X),on_left(O1,O2),type(O1,agent),type(O2,door).
0.291:right_go_get_key(X):-not_have_key(X),on_left(O1,O2),type(O1,agent),
                           type(O2,key).
0.562:right_go_to_door(X):-have_key(X),on_left(O1,O2),type(O1,agent),type(O2,door).

#3Fishes
0.779:right_to_eat(X):-is_bigger_than(O1,O2),on_left(O2,O1),type(O1,agent),
                       type(O2,fish).
0.445:down_to_dodge(X):-is_bigger_than(O2,O1),on_left(O2,O1),type(O1,agent),
                       type(O2,fish).
0.579:down_to_eat(X):-high_level(O1,O2),is_smaller_than(O2,O1),type(O1,agent),
                       type(O2,fish).
0.699:up_to_dodge(X):-closeby(O2,O1),is_smaller_than(O1,O2),low_level(O2,O1),
                       type(O1,agent),type(O2,fish).
0.601:up_to_eat(X):-is_bigger_than(O2,O1),on_left(O2,O1),type(O1,agent),
                       type(O2,fish).
0.581:left_to_eat(X):-closeby(O1,O2),on_right(O1,O2),type(O1,agent),type(O2,fish).


# Loot
0.844:up_to_door(X):-close(O1,O2),have_key(O2),on_top(O2,O1),type(O1,agent),
                       type(O2,door).
0.268:right_to_key(X):-close(O1,O2),on_right(O2,O1),type(O1,agent),type(O2,key).
0.732:right_to_door(X):-close(O1,O2),have_key(O2),on_left(O1,O2),type(O1,agent),
                       type(O2,door).
0.508:up_to_key(X):-close(O1,O2),on_top(O2,O1),type(O1,agent),type(O2,key).
0.995:left_to_door(X):-close(O1,O2),have_key(O2),on_left(O2,O1),type(O1,agent),
                       type(O2,door).
0.414:down_to_key(X):-close(O1,O2),on_top(O1,O2),type(O1,agent),type(O2,key).
0.992:down_to_door(X):-close(O1,O2),have_key(O2),on_top(O1,O2),type(O1,agent),
                       type(O2,door).
0.447:left_to_key(X):-close(O1,O2),on_left(O2,O1),type(O1,agent),type(O2,key).
```

Figure 8: **NUDGE produces an interpretable policy as set of weighted rules.** Weighted action rules discovered by NUDGE in the each logic environment.

D Hyperparameters and rules sets

D.1 Hyperparameters

We here provide the hyperparameters used in our experiments. We set the clip parameter $\epsilon_{clip} = 0.2$, the discount factor $\gamma = 0.99$. We use the Adam optimizer, with $1e-3$ as actor learning rate, $3e-4$ as critic learning rate. The episode length is 500 timesteps. The policy is updated every 1000 steps We train every algorithm for $800k$ steps on each environment, apart from neural PPO, that needed $5M$ steps on Loot. We use an epsilon greedy strategy with $\epsilon = max(e^{\frac{-episode}{500}}, 0.02)$.

D.2 Rules set

All the rules set $\mathcal{C}$ of the different NUDGE and logic agents are available at https://anonymous.4open.science/r/LogicRL-C43B in the folder nsfr/nsfr/data/lang.

E Details of Differentiable Forward Reasoning

We provide details of differentiable forward reasoning used in NUDGE. We denote a valuation vector at time step t as $\mathbf{v}^{(t)} \in [0,1]^G$. We also denote the i-th element of vector $\mathbf{x}$ by $\mathbf{x}[i]$, and the (i,j)-th element of matrix $\mathbf{X}$ by $\mathbf{X}[i,j]$. The same applies to higher dimensional tensors.

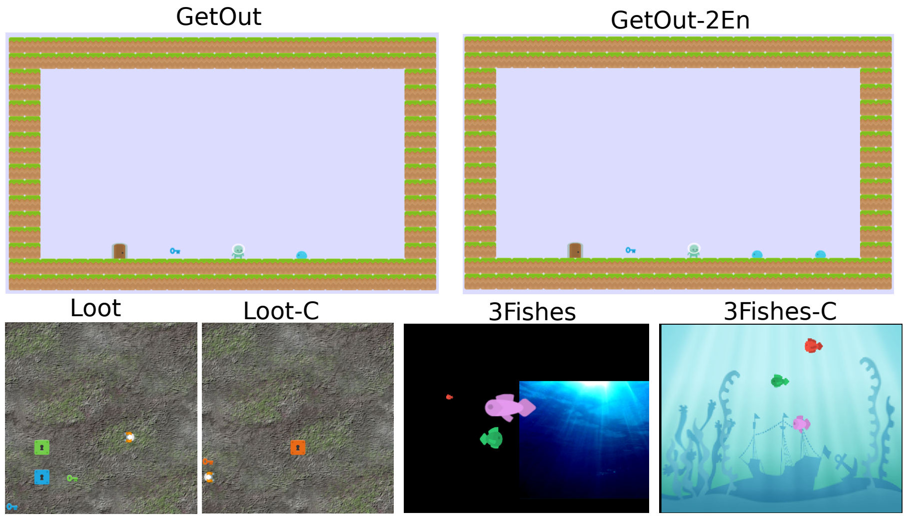

Figure 9: Pictures of our environments (**GetOut**, **Loot** and **3Fishes**) and their variations (**GetOut-2En**, **Loot-C** and **3Fishes-C**). All these environments can provide object-centric state descriptions (instead of pixel-based states).

E.1 Differentiable Forward Reasoning

We compose the reasoning function $f^{reason}_{(\mathcal{C},\mathbf{W})} : [0,1]^G \rightarrow [0,1]^{G_A}$, which takes the initial valuation vector and returns valuation vector for induced action atoms. We describe each step in detail.

(Step 1) Encode Logic Programs to Tensors. To achieve differentiable forward reasoning, each action rule is encoded to a tensor representation. Let S be the number of the maximum number of substitutions for existentially quantified variables in $\mathcal{C}$, and L be the maximum length of the body of rules in $\mathcal{C}$. Each action rule $C_i \in \mathcal{C}$ is encoded to a tensor $\mathbf{I}_i \in \mathbb{N}^{G \times S \times L}$, which contain the indices of body atoms. Intuitively, $\mathbf{I}_i[j,k,l]$ is the index of the l-th fact (subgoal) in the body of the i-th rule to derive the j-th fact with the k-th substitution for existentially quantified variables.

For example. let $R_0 = \mathtt{jump(agent)}\text{:-}\mathtt{type(O1, agent)}, \mathtt{type(O2, enemy)}, \mathtt{closeby(O1, O2)} \in \mathcal{C}$ and $F_2 = \mathtt{jump(agent)} \in \mathcal{G}$, and we assume that constants for objects are $\{\mathtt{obj1}, \mathtt{obj2}\}$. R_0 has existentially quantified variables O1 and O2 on the body, so we obtain ground rules by substituting constants. By considering the possible substitutions for O1 and O2, namely $\{\mathtt{O1/obj1}, \mathtt{O2/obj2}\}$ and $\{\mathtt{O1/obj2}, \mathtt{O2/obj1}\}$, we have *two* ground rules, as shown in top of Table 3. Bottom rows of Table 3 shows elements of tensor $\mathbf{I}_{0,:,0,:}$ and $\mathbf{I}_{0,:,1,:}$. Facts $\mathcal{G}$ and the indices are represented on the upper rows in the table. For example, $\mathbf{I}_{0,2,0,:} = [3,6,7]$ because R_0 entails $\mathtt{jump(agent)}$ with the first ($k = 0$) substitution $\tau = \{\mathtt{O1/obj1}, \mathtt{O2/obj2}\}$. Then the subgoal atoms are $\{\mathtt{type(obj1, agent)}, \mathtt{type(obj2, enemy)}, \mathtt{closeby(obj1, obj2)}\}$, which have indices $[3,6,7]$, respectively. The atoms which have a different predicate, e.g., $\mathtt{closeby(obj1, obj2)}$, will never be entailed by clause R_0. Therefore, the corresponding values are filled with 0, which represents the index of the *false* atom.

(Step 2) Assign Rule Weights. We assign weights to compose the policy with several action rules as follows: (i) We fix the target programs' size as M, *i.e.* , where we try to find a policy with M action rules. (ii) We introduce C-dim weights $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_M]$. (iii) We take the *softmax* of each weight vector $\mathbf{w}_j \in \mathbf{W}$ and softly choose M action rules out of C action rules to compose the policy.

(Step 3) Perform Differentiable Inference. We compute 1-step forward reasoning using weighted action rules, then we recursively perform reasoning to compute T-step reasoning.

[(i) Reasoning using an action rule] First, for each action rule $C_i \in \mathcal{C}$, we evaluate body atoms for different grounding of C_i by computing $b^{(t)}_{i,j,k} \in [0,1]$:

$$b^{(t)}_{i,j,k} = \prod_{1 \leq l \leq L} \mathbf{gather}(\mathbf{v}^{(t)}, \mathbf{I}_i)[j,k,l] \tag{8}$$

$(k=0)$ $\quad$ $\texttt{jump(agent):-type(obj1, agent), type(obj2, enemy), closeby(obj1, obj2).}$
$(k=1)$ $\quad$ $\texttt{jump(agent):-type(obj2, agent), type(obj1, enemy), closeby(obj2, obj1).}$

j	0	1	2	3	4	5
$\mathcal{G}$	$\perp$	$\top$	jump(agent)	type(obj1, agent)	type(obj2, agent)	type(obj1, enemy)
$\mathbf{I}_{0,j,0,:}$	$[0,0,0]$	$[1,1,1]$	$[3,6,7]$	$[0,0,0]$	$[0,0,0]$	$[0,0,0]$
$\mathbf{I}_{0,j,1,:}$	$[0,0,0]$	$[1,1,1]$	$[4,5,8]$	$[0,0,0]$	$[0,0,0]$	$[0,0,0]$

j	6	7	8	...
$\mathcal{G}$	type(obj2, enemy)	closeby(obj1, obj2)	closeby(obj2, obj1)	...
$\mathbf{I}_{0,j,0,:}$	$[0,0,0]$	$[0,0,0]$	$[0,0,0]$	...
$\mathbf{I}_{0,j,1,:}$	$[0,0,0]$	$[0,0,0]$	$[0,0,0]$	...

Table 3: Example of ground rules (top) and elements in the index tensor (bottom). Each fact has its index, and the index tensor contains the indices of the facts to compute forward inferences.

where $\mathbf{gather} : [0,1]^G \times \mathbb{N}^{G\times S\times L} \rightarrow [0,1]^{G\times S\times L}$ is:

$$\mathbf{gather}(\mathbf{x}, \mathbf{Y})[j,k,l] = \mathbf{x}[\mathbf{Y}[j,k,l]]. \tag{9}$$

The $\mathbf{gather}$ function replaces the indices of the body state atoms by the current valuation values in $\mathbf{v}^{(t)}$. To take logical *and* across the subgoals in the body, we take the product across valuations. $b_{i,j,k}^{(t)}$ represents the valuation of body atoms for i-th rule using k-th substitution for the existentially quantified variables to deduce j-th fact at time t.

Now we take logical *or* softly to combine all of the different grounding for C_i by computing $c_{i,j}^{(t)} \in [0,1]$:

$$c_{i,j}^{(t)} = \mathit{softor}^\gamma(b_{i,j,1}^{(t)}, \ldots, b_{i,j,S}^{(t)}) \tag{10}$$

where softor^γ is a smooth logical *or* function:

$$\mathit{softor}^\gamma(x_1, \ldots, x_n) = \gamma \log \sum_{1\leq i\leq n} \exp(x_i/\gamma), \tag{11}$$

where $\gamma > 0$ is a smooth parameter. Eq. 11 is an approximation of the *max* function over probabilistic values based on the *log-sum-exp* approach [Cuturi and Blondel, 2017].

[(ii) Combine results from different action rules] Now we apply different action rules using the assigned weights by computing $h_{j,m}^{(t)} \in [0,1]$:

$$h_{j,m}^{(t)} = \sum_{1\leq i\leq C} w_{m,i}^* \cdot c_{i,j}^{(t)}, \tag{12}$$

where $w_{m,i}^* = \exp(w_{m,i})/\sum_{i'}\exp(w_{m,i'})$, and $w_{m,i} = \mathbf{w}_m[i]$. Note that $w_{m,i}^*$ is interpreted as a probability that action rule $C_i \in \mathcal{C}$ is the m-th component of the policy. Now we complete the 1-step forward reasoning by combining the results from different weights:

$$r_j^{(t)} = \mathit{softor}^\gamma(h_{j,1}^{(t)}, \ldots, h_{j,M}^{(t)}). \tag{13}$$

Taking softor^γ means that we compose the policy using M softly chosen action rules out of C candidates of rules.

[(iii) Multi-step reasoning] We perform T-step forward reasoning by computing $r_j^{(t)}$ recursively for T times: $v_j^{(t+1)} = \mathit{softor}^\gamma(r_j^{(t)}, v_j^{(t)})$. Finally, we compute $\mathbf{v}^{(T)} \in [0,1]^G$ and returns $\mathbf{v}_A \in [0,1]^{G_A}$ by extracting only output for action atoms from $\mathbf{v}^{(T)}$. The whole reasoning computation Eq. 8-13 can be implemented using only efficient tensor operations. See App. E.2 for a detailed description.

E.2 Implementation Details

Here we provide implementational details of the differentiable forward reasoning. The whole reasoning computations in NUDGE can be implemented as a neural network that performs forward reasoning and can efficiently process a batch of examples in parallel on GPUs, which is a non-trivial function of logical reasoners.

Each clause $C_i \in \mathcal{C}$ is compiled into a differentiable function that performs forward reasoning using the tensor. The clause function is computed as:

$$\mathbf{C}_i^{(t)} = \mathit{softor}_3^{\gamma}\Big(\mathit{prod}_2\big(\mathit{gather}_1(\tilde{\mathbf{V}}^{(t)}, \tilde{\mathbf{I}})\big)\Big), \tag{14}$$

where $\mathit{gather}_1(\mathbf{X}, \mathbf{Y})_{i,j,k,l} = \mathbf{X}_{i,\mathbf{Y}_{i,j,k,l},k,l}$ [2] obtains valuations for body atoms of the clause C_i from the valuation tensor and the index tensor. prod_2 returns the product along dimension 2, *i.e.* the product of valuations of body atoms for each grounding of C_i. The softor^{γ} function is applied along dimension 3, on all the grounding (or possible substitutions) of C_i.

$\mathit{softor}_d^{\gamma}$ is a function for taking logical *or* softly along dimension d:

$$\mathit{softor}_d^{\gamma}(\mathbf{X}) = \gamma \log\big(\mathit{sum}_d \exp(\mathbf{X}/\gamma)\big), \tag{15}$$

where $\gamma > 0$ is a smoothing parameter, sum_d is the sum function along dimension d. The results from each clause $\mathbf{C}_i^t \in \mathbb{R}^{B \times G}$ is stacked into tensor $\mathbf{C}^{(t)} \in \mathbb{R}^{C \times B \times G}$.

Finally, the T-time step inference is computed by amalgamating the inference results recursively. We take the $\mathrm{softmax}$ of the clause weights, $\mathbf{W} \in \mathbb{R}^{M \times C}$, and softly choose M clauses out of C clauses to compose the logic program:

$$\mathbf{W}^* = \mathit{softmax}_1(\mathbf{W}). \tag{16}$$

where $\mathit{softmax}_1$ is a softmax function over dimension 1. The clause weights $\mathbf{W}^* \in \mathbb{R}^{M \times C}$ and the output of the clause function $\mathbf{C}^{(t)} \in \mathbb{R}^{C \times B \times G}$ are expanded (via copy) to the same shape $\tilde{\mathbf{W}}^*, \tilde{\mathbf{C}}^{(t)} \in \mathbb{R}^{M \times C \times B \times G}$. The tensor $\mathbf{H}^{(t)} \in \mathbb{R}^{M \times B \times G}$ is computes as

$$\mathbf{H}^{(t)} = \mathit{sum}_1(\tilde{\mathbf{W}}^* \odot \tilde{\mathbf{C}}), \tag{17}$$

where $\odot$ is element-wise multiplication. Each value $\mathbf{H}_{i,j,k}^{(t)}$ represents the weight of k-th ground atom using i-th clause weights for the j-th example in the batch. Finally, we compute tensor $\mathbf{R}^{(t)} \in \mathbb{R}^{B \times G}$ corresponding to the fact that logic program is a set of clauses:

$$\mathbf{R}^{(t)} = \mathit{softor}_0^{\gamma}(\mathbf{H}^{(t)}). \tag{18}$$

With r the 1-step forward-chaining reasoning function:

$$r(\mathbf{V}^{(t)}; \mathbf{I}, \mathbf{W}) = \mathbf{R}^{(t)}, \tag{19}$$

we compute the T-step reasoning using:

$$\mathbf{V}^{(t+1)} = \mathit{softor}_1^{\gamma}\Big(\mathit{stack}_1\big(\mathbf{V}^{(t)}, r(\mathbf{V}^{(t)}; \mathbf{I}, \mathbf{W})\big)\Big), \tag{20}$$

where $\mathbf{I} \in \mathbb{N}^{C \times G \times S \times L}$ is a precomputed index tensor, and $\mathbf{W} \in \mathbb{R}^{M \times C}$ is clause weights. After T-step reasoning, the probabilities over action atoms $\mathcal{G}_A$ are extracted from $\mathbf{V}^{(T)}$ as $\mathbf{V}_A \in [0, 1]^{B \times G_A}$.

[2] done with pytorch.org/docs/torch.gather