EXPLORER: Exploration-guided Reasoning
for Textual Reinforcement Learning

Kinjal Basu,[1] Keerthiram Murugesan,[1] Subhajit Chaudhury,[1] Murray Campbell,[1]
Kartik Talamadupula,[2] and Tim Klinger,[1]

[1] IBM Research   [2] Symbl.ai

{kinjal.basu, keerthiram.murugesan, subhajit}@ibm.com,
kartik.t@symbl.ai, {mcam, tklinger}@us.ibm.com

Abstract

Text-based games (TBGs) have emerged as an important collection of NLP tasks, requiring reinforcement learning (RL) agents to combine natural language understanding with reasoning. A key challenge for agents attempting to solve such tasks is to generalize across multiple games and demonstrate good performance on both seen and unseen objects. Purely deep-RL-based approaches may perform well on seen objects; however, they fail to showcase the same performance on unseen objects. Commonsense-infused deep-RL agents may work better on unseen data; unfortunately, their policies are often not interpretable or easily transferable. To tackle these issues, in this paper, we present EXPLORER[1] which is an exploration-guided reasoning agent for textual reinforcement learning. EXPLORER is neuro-symbolic in nature, as it relies on a neural module for exploration and a symbolic module for exploitation. It can also learn generalized symbolic policies and perform well over unseen data. Our experiments show that EXPLORER outperforms the baseline agents on Text-World cooking (TW-Cooking) and Text-World Commonsense (TWC) games.

1 Introduction

Natural language plays a crucial role in human intelligence and cognition. To study and evaluate the process of language-informed sequential decision-making in AI agents, text-based games (TBGs) have emerged as important simulation environments, where the states and actions are usually described in natural language. To solve game instances, an agent needs to master both natural language processing (NLP) and reinforcement learning (RL). At a high level, existing RL agents for TBGs can be classified into two classes: (a) rule-based agents, and (b) neural agents. Rule-based agents such as NAIL (Hausknecht et al., 2019)
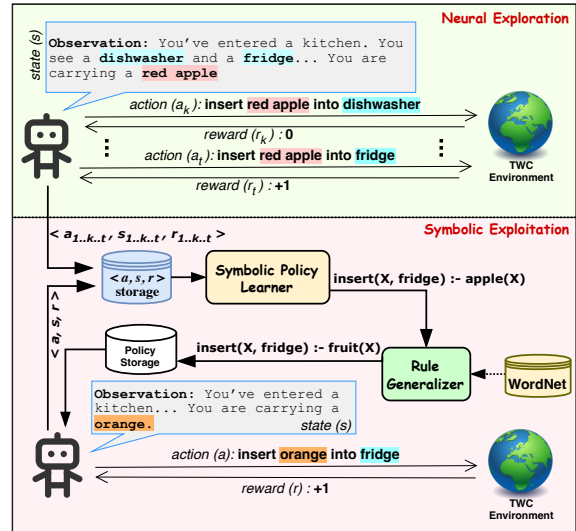

Figure 1: An overview of the EXPLORER agent's dataflow on a TWC game. In EXPLORER, the neural module is responsible for *exploration* and collects <*action*, *state*, *reward*> pairs, whereas the symbolic module learns the rules and does the *exploitation* using commonsense knowledge from WordNet.

rely heavily on prior predefined knowledge. This makes them less flexible and adaptable. To overcome the challenges of rule-based agents, in recent years, with the advent of new deep learning techniques, significant progress has been made on neural agents (Narasimhan et al., 2015; Adhikari et al., 2020b). However, these frameworks also suffer from a number of shortcomings. First, from deep learning, they inherit the need for very large training sets, which entails that they learn slowly. Second, they are brittle in the sense that a trained network may show good performance with the entities that are seen in the training instances, yet it performs very poorly in a very similar environment with unseen entities. Additionally, the policies learned by these neural RL agents are not interpretable (human-readable).

---

[1]Code available at: https://github.com/kinjalbasu/explorer

In this paper, we introduce **EXPLORER** for TBGs that utilizes the positive aspects of both neural and symbolic agents. The EXPLORER is based on two modules - neural and symbolic, where the neural module is mainly responsible for *exploration* and the symbolic module does the *exploitation*. An overview of the EXPLORER agent can be found in Figure 1. A key advantage of EXPLORER is that it has a scalable design that can integrate any neural module and can build the symbolic module upon it. For the symbolic module, instead of using predefined prior knowledge, EXPLORER learns its symbolic policies by leveraging reward and action pairs while playing the game. These policies are represented using a declarative logic programming paradigm — Answer Set Programming (ASP) (Lifschitz, 2019), which allows the policies to be interpretable and explainable. Due to its non-monotonic nature and efficient knowledge representation ability, ASP has proven its efficiency in NLP research (Basu et al., 2020, 2021; Pendharkar et al., 2022; Zeng et al., 2024); Commonsense reasoning research (Gupta et al., 2023; Kothawade et al., 2021); and NLP + RL research (Lyu et al., 2019; Basu et al., 2022b; Sridharan et al., 2017; Mitra and Baral, 2015; Yang et al., 2018). We believe non-monotonic reasoning (NMR) (Gelfond and Lifschitz, 1988; Reiter, 1988) is a crucial capability in partially observable worlds, as the agent's beliefs can change in the presence of new information and examples. Importantly, with the help of an exception learner (illustrated in Section 3.2), EXPLORER learns the symbolic policies as default theories so that the agent can perform NMR, and the policies remain consistent with the agent's findings.

After learning the symbolic policies, EXPLORER can *lift* or variablize the rules using WordNet (Miller, 1995) to generalize them. By generalizing the symbolic policies, we find that EXPLORER overcomes the challenge of getting poor performance over unseen entities or out-of-distribution (OOD) test sets, as the unseen objects are similar in nature to the training objects and occur under the same class in WordNet.

Figure 2 illustrates the components of our neuro-symbolic architecture and shows an overview of the agent's decision-making process.

We have used TW-cooking to verify our approach and then performed a comprehensive evaluation of EXPLORER on TWC games. To showcase the scalability aspects of EXPLORER, we
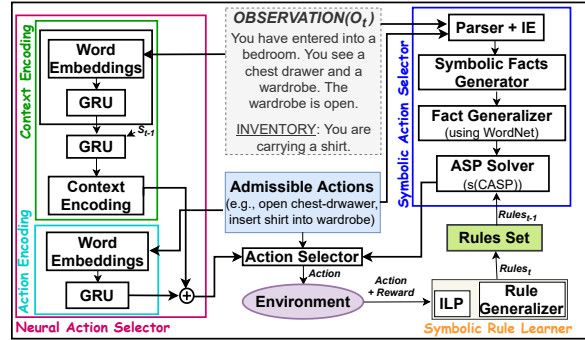

Figure 2: Overview of EXPLORER's decision-making at any given time step. The Hybrid Neuro-Symbolic architecture mainly consists of 5 modules - (a) Context Encoder encodes the observation to dynamic context, (b) Action Encoder encodes the admissible actions, (c) Neural Action Selector combines (a) and (b) with $\oplus$ operator, (d) Symbolic Action Selector returns a set of candidate actions, and (e) Symbolic Rule Learner uses ILP and WordNet-based rule generalization to generate symbolic rules.

have done comparative studies with other SOTA neural and neuro-symbolic models, and the empirical results demonstrate that EXPLORER outplays others by achieving better generalization over unseen entities. Due to the neuro-symbolic nature of EXPLORER, we are also able to perform detailed qualitative studies of the policies (illustrated in section - 5.3).

The main contributions of this paper are: (1) we present EXPLORER for TBGs that outperforms existing models in terms of steps and scores; (2) we discuss the importance of non-monotonic reasoning in partially observable worlds; (3) we demonstrate how default theories can be learned with exceptions in an online manner for TBGs; and (4) we provide a novel information-gain based rule generalization algorithm that leverages WordNet.

2 Background

Text-based Reinforcement Learning: TBGs provide a challenging environment where an agent can observe the current state of the game and act in the world using only the modality of text. The agent perceives the state of the game only through natural language observations. Hence, TBGs can be modeled as a Partially Observable Markov Decision Process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r)$, where $\mathcal{S}$ is the set of states of the game, $\mathcal{A}$ is the natural language action space, $\mathcal{O}$ is the set of textual observations describing the current state, $\mathcal{T}$ are the conditional transition probabilities from one state

to another, $\mathcal{E}$ are the conditional observation probabilities, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a scalar reward function, which maps a state-action pair to the reward received by the agent.

Inductive Logic Programming (ILP): ILP is a machine learning technique where the learned model is in the form of logic programming rules (Horn Clauses) that are comprehensible to humans. It allows the background knowledge to be incrementally extended without requiring the entire model to be re-learned. Additionally, the comprehensibility of symbolic rules makes it easier for users to understand and verify induced models and even edit them. Details can be found in the work of Muggleton and De Raedt (1994).

Answer Set Programming (ASP): An answer set program is a collection of rules of the form:

$$l_0 \leftarrow l_1, \ ... \ , \ l_m, \ not \, l_{m+1}, \ ... \ , \ not \, l_n.$$

Classical logic denotes each l_i is a literal (Gelfond and Kahl, 2014). ASP supports *negation as failure* (Gelfond and Kahl, 2014), allowing it to elegantly model common sense reasoning, default rules with exceptions, etc.

s(CASP) Engine: For this work, we have used s(CASP) ASP solver to predict an action. s(CASP) (Arias et al., 2018) is a query-driven, goal-directed implementation of ASP that includes constraint solving over reals. Goal-directed execution of s(CASP) is indispensable for automating commonsense reasoning, as traditional grounding and SAT-solver based implementations of ASP may not be scalable. There are three major advantages of using the s(CASP) system: (i) s(CASP) does not ground the program, which makes our framework scalable, (ii) it only explores the parts of the knowledge base that are needed to answer a query, and (iii) it provides natural language justification (proof tree) for an answer (Arias et al., 2020).

3 Symbolic Policy Learner

Deep reinforcement learning (DRL) has experienced great success by learning directly from high-dimensional sensory inputs, yet it suffers from a lack of interpretability. Interpretability of an agent's action is of utmost importance in sequential decision-making problems, as it increases the transparency of black-box-style agents; it also helps RL researchers understand the high-level behavior of the system better. To make a system interpretable, one of the most widely used

approaches is learning the agent's policies symbolically. In our work, EXPLORER learns these symbolic policies in the form of logical rules represented in the ASP. An example of such a rule is `- insert(X, fridge) :- apple(X)` which can be translated as *"X is insertable into a fridge if X is an apple"* [2]. These learned ASP rules not only provide a better understanding of the system's functionality but can also be used to predict the agent's next action using an ASP solver. EXPLORER learns the rules iteratively (in an online manner) and applies the rules to predict an action in collaboration with the neural module. Our results show that this approach is very effective in terms of performance and interpretability.

Partial Observability and Non-Monotonic Reasoning: EXPLORER works in a partially observable environment, where it needs to predict an action based on its prior knowledge. If EXPLORER fails, then it learns something new that will be applied in the next episode. The reasoning approach of EXPLORER is non-monotonic in nature: that is, what it believes currently may become false in the future with new evidence. We can model this using a non-monotonic logic programming paradigm that supports *default rules* and *exception to defaults* (Gelfond and Kahl, 2014). In this work, the belief of EXPLORER has been represented as an Answer Set Program in the form of *default rules* with *exceptions*. With the help of Inductive Logic Programming (ILP) (see Section 3.1) and Exception learner (see Section 3.2), these rules are learned by EXPLORER after each episode and then applied in the following episode. The agent uses an ASP solver to predict actions by utilizing the observation and the rules. Based on the outcome after applying the rules, the learned policies are updated with the exception (if needed), and new rules are learned as needed.

3.1 Learning Symbolic Policy using ILP

Data Collection: To apply an ILP algorithm, first, EXPLORER needs to collect the *State*, *Action*, and *Reward* pairs while exploring the text-based environment. In a TBG, the two main components of the state are the state description and the inventory information of the agent. The entities present in the environment are extracted by parsing the state description using the spaCy library, and only storing the noun phrases (e.g., fridge, apple, banana,

---

[2]For ease of use, we retain action names as the predicate names; however, they are interpreted normally as adjectives.
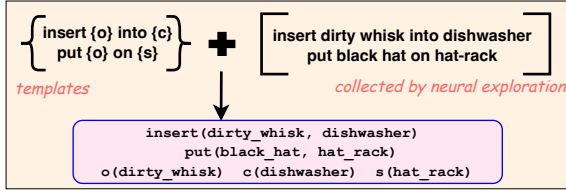
Figure 3: Entity extraction using Action Template

Figure 4: ILP Rule Learning Example

etc.) in predicate form. We also extract the inventory information in a similar way. At each step of the game, the game environment generates a set of admissible actions, one among them being the best; as well as action templates (e.g., *"insert* O *into* S*"*, where O and S are entity types) which are predefined for the agent before the game starts. By processing these templates over the admissible actions, EXPLORER can easily extract the type of each entity present in the environment and then convert them to predicates. Figure 3 illustrates an instance of a predicate generation process. Along with this *State* description, EXPLORER also stores the taken *Action* and the *Reward* information at each step.

Data Preparation: To learn the rules, an ILP algorithm requires three things - the *goal*, the *predicate list*, and the *examples*. The *goal* is the concept that the ILP algorithm is going to learn by exploring the examples. The *predicates* give the explanation to a concept. In the learned theory formulated as logical rules, *goal* is the head and the *predicate list* gives the domain space for the body clauses. The *examples* are the set of positive and negative scenarios that are collected by the agent while playing.

Execution and Policy Learning: In our work, we have mainly focused on learning the hypothesis for the rewarded actions; however, we also apply reward shaping to learn important preceding actions (e.g., *open fridge* might not have any reward, although it is important to *take an item from fridge* and that has a reward). In both the TW-Cooking domain and TextWorld Commonsense (TWC), the action predicates mostly have one or two arguments (e.g., *open* fridge, *insert* cheese *in* fridge, etc.). In the one-argument setting, the *action* becomes the ILP goal and the examples are collected based on the argument. In the two-argument setting, we fix the second argument with the action and collect examples based on the first argument. The goal will hence be in the form of *<action_(second_argument) >*. We split the examples (i.e., state, entity types, inventory information
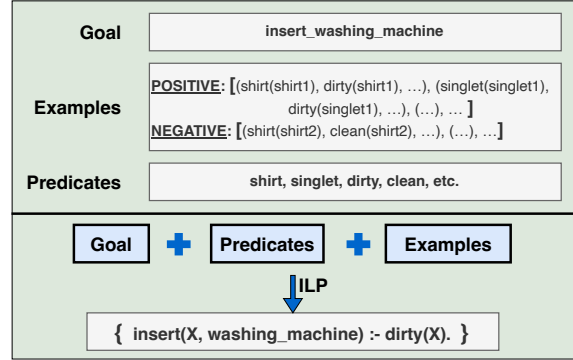
in predicate form) based on the stored rewards (positive and zero/negative). We use entity identifiers to identify each entity separately; this is important when there are two or more instances of the same entity in the environment with different features (e.g., *red apple* and *rotten apple*). Additionally, EXPLORER creates the predicate list by extracting the predicate names from the examples. After obtaining the goal, predicate list, and the example, the agent runs the ILP algorithm to learn the hypothesis, followed by simple string post-processing to obtain a hypothesis in the below form:

```
action( X , entity) <- feature(X).
action( X ) <- feature(X).
```

Figure 4 elaborates the ILP data preparation procedure along with an example of a learned rule.

3.2 Exception Learning

As EXPLORER does online learning, the quality of the initial rules is quite low; this gradually improves with more training. The key improvement achieved by EXPLORER is through exception learning, where an exception clause is added to the rule's body using Negation as Failure (NAF). This makes the rules more flexible and able to handle scenarios where information is missing. The agent learns these exceptions by trying the rules and not receiving rewards. For example, in TWC, the agent may learn the rule that - *apple goes to the fridge*, but fail when it tries to apply the rule to a *rotten apple*. It then learns that the feature *rotten* is an exception to the previously learned rule. This can be represented as:

```
insert(X , fridge) <- apple(X), not ab(X).
ab(X) <- rotten(X).
```

It is important to keep in mind that the number of examples covered by the exception is always fewer than the number of examples covered by
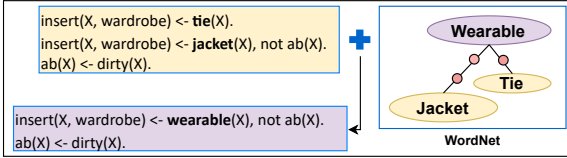
Figure 5: Example of Rule Generalization

the defaults. This constraint has been included in EXPLORER's exception learning module.

4 Rule Generalization

Importance of Rule Generalization: An ideal RL agent should not only perform well on entities it has seen but also on unseen entities or out-of-distribution (OOD) data. To accomplish this, policy generalization is a crucial feature that an ideal RL agent should have. To verify this, we used EXPLORER without generalization on the TW-Cooking domain, where it performs well, however, it struggles on the TWC games. TWC games are designed to test agents on OOD entities that were not seen during training but are similar to the training data. As a result, the policies learned as logic rules will not work on unseen objects.

For example, the rule for *apple* (e.g., `insert(X, fridge) <- apple(X). )` cannot work on another fruit such as *orange.* To tackle this, we lift the learned policies using WordNet's (Miller, 1995) hypernym-hyponym relations to get the generalized rules (illustration in Figure 5). Motivation comes from the way humans perform tasks. For example, if we know a *dirty shirt* goes to the *washing machine* and we have seen a *dirty pant*, we would put the *dirty pant* into the *washing machine* as both are of type *clothes* and *dirty*.

Excessive Generalization is Bad: On one hand, generalization results in better policies to work with unseen entities; however, too much generalization leads to a drastic increment in false-positive results. To keep the balance, EXPLORER should know how much generalization is good. For an example, "*apple* is a *fruit*", "*fruits* are *part of a plant*", and "*plants* are *living thing*". Now, if we apply the same rule that explains a property of an *apple* to all *living things*, the generalization will have gone too far. So, to solve this, we have proposed a novel approach described in Section - 4.1.

4.1 Dynamic Rule Generalization

In this paper, we introduce a novel algorithm to dynamically generate the generalized rules explor-

ing the hypernym relations from WordNet (WN). The algorithm is based on information gain calculated using the entropy of the positive and negative set of examples (collected by EXPLORER). The illustration of the process is given in the Algorithm 1. The algorithm takes the collected set of examples and returns the generalized rules set. First, similar to the ILP data preparation procedure, the *goals* are extracted from the examples. For each *goal*, examples are split into two sets - E^+ and E^-. Next, the hypernyms are extracted using the hypernym-hyponym relations of the WordNet ontology. The combined set of hypernyms from (E^+, E^-) gives the *body* predicates for the generalized rules. Similar to the ILP (discussed above) the *goal* will be the *head* of a generalized rule. Next, the best-generalized rules are generated by calculating the max information gain between the hypernyms. Information gain for a given clause is calculated using the below formula (Mitchell, 1997) —

$$IG(R, h) = total * (log_2 \frac{p_1}{p_1 + n_1} - log_2 \frac{p_0}{p_0 + n_0})$$

where h is the candidate hypernym predicate to add to the rule R, p_0 is the number of positive examples implied by the rule R, n_0 is the number of negative examples implied by the rule R, p_1 is the number of positive examples implied by the rule $R + h$, n_1 is the number of negative examples implied by the rule $R + h$, $total$ is the number of positive examples implied by R also covered by $R + h$. Finally, it collects all the generalized rules set and returns. It is important to mention that this algorithm only learns the generalized rules which

---

Algorithm 1 Generalized Rule Learner

Input: *E*: Examples (*States, Actions, and Rewards*)
Output: R_G: Generalized Rules Set
1: **procedure** GETGENERALIZEDRULES(E)
2:     $R_G \leftarrow \{\}$ ▷ initialization
3:     $Goals \leftarrow getGoals(E)$ ▷ get the list of goals similar to the ILP data preparation (described above)
4:     **for each** $g \in Goals$ **do**
5:         $E_g \leftarrow getExamples(E, g)$
6:         $(E_g^+, E_g^-) \leftarrow splitByRewards(E_g)$
7:         $(Hyp_g^+, Hyp_g^-) \leftarrow extractHypernyms(E_g^+, E_g^-)$

            ▷ get the hypernyms from WordNet
8:         $r_g \leftarrow getBestGen(E_g^+, E_g^-, Hyp_g^+, Hyp_g^-)$
                ▷ uses entropy based information gain formula
9:         $R_G \leftarrow R_G \cup r_g$
10:     **end for**
11:     **return** R_G
12: **end procedure**

			Easy		Medium		Hard	
			#steps	N. Score	#steps	N. Score	#steps	N. Score
IN	Text Only		15.12 ± 1.95	0.91 ± 0.03	33.17 ± 2.76	0.83 ± 0.04	47.68 ± 2.43	0.6 ± 0.05
	EXPLORER-w/o-GEN		17.39 ± 3.01	0.93 ± 0.04	46.7 ± 2.14	0.42 ± 0.12	37.66 ± 0.93	0.88 ± 0.01
	EXPLORER	Exhaustive	12.86 ± 3.04	0.91 ± 0.04	29.9 ± 3.16	0.65 ± 0.06	30.44 ± 0.87	0.95 ± 0.03
		IG (Hyp. Lvl. 2)	10.59 ± 1.3	0.95 ± 0.02	22.57 ± 1.04	0.77 ± 0.07	30.46 ± 0.74	0.87 ± 0.01
		IG (Hyp. Lvl. 3)	9.55 ± 2.34	0.96 ± 0.02	25.34 ± 2.86	0.76 ± 0.03	33.54 ± 1.47	0.91 ± 0.03
OUT	Text Only		16.66 ± 1.74	0.92 ± 0.03	37.3 ± 3.45	0.73 ± 0.06	50.00 ± 0.0	0.3 ± 0.04
	EXPLORER-w/o-GEN		21.19 ± 0.87	0.84 ± 0.06	46.36 ± 1.52	0.42 ± 0.08	44.25 ± 0.42	0.63 ± 0.01
	EXPLORER	Exhaustive	14.65 ± 2.18	0.91 ± 0.05	37.07 ± 2.09	0.63 ± 0.06	41.52 ± 1.12	0.83 ± 0.02
		IG (Hyp. Lvl. 2)	15.08 ± 1.2	0.91 ± 0.02	40.63 ± 3.03	0.57 ± 0.06	42.18 ± 0.66	0.79 ± 0.01
		IG (Hyp. Lvl. 3)	12.72 ± 1.22	0.92 ± 0.02	37.38 ± 3.09	0.64 ± 0.09	43.16 ± 2.83	0.78 ± 0.03

Table 1: TWC performance comparison results for within distribution (IN) and out-of-distribution (OUT) games

are used in addition to the rules learned by ILP and exception learning (discussed in section 3).

5 Experiments and Results

5.1 Dataset

In our work, we want to show that if an RL agent uses symbolic and neural reasoning in tandem, where the neural module is mainly responsible for exploration and the symbolic component for exploitation, then the performance of that agent increases drastically in text-based games. At first, we verify our approach with TW-Cooking domain (Adhikari et al., 2020a), where we have used levels 1-4 from the GATA dataset[3] for testing. As the name suggests, this game suit is about collecting various cooking ingredients and preparing a meal following an in-game recipe.

To showcase the importance of generalization, we have tested our EXPLORER agent on TWC games with OOD data. Here, the goal is to tidy up the house by putting objects in their commonsense locations. With the help of TWC framework (Murugesan et al., 2021a), we have generated a set of games with 3 different difficulty levels - (i) *easy level:* that contains 1 room with 1 to 3 objects; (ii) *medium level:* that contains 1 or 2 rooms with 4 or 5 objects; and (iii) *hard level:* a mix of games with a high number of objects (6 or 7 objects in 1 or 2 rooms) or a high number of rooms (3 or 4 rooms containing 4 or 5 objects).

We chose TW-Cooking and TWC games as our test-bed because these are benchmark datasets for evaluating neuro-symbolic agents in text-based games (Chaudhury et al., 2021, 2023; Wang et al., 2022; Kimura et al., 2021; Basu et al., 2022a). Also, these environments require the agents to exhibit skills such as exploration, planning, reasoning, and OOD generalization, which makes them ideal environments to evaluate EXPLORER.

5.2 Experiments

To explain EXPLORER works better than a neural-only agent, we have selected two neural baseline models for each of our datasets (TWC and TW-Cooking) and compared them with **EXPLORER**. In our evaluation, for both the datasets, we have used LSTM-A2C (Narasimhan et al., 2015) as the **Text-Only** agent, which uses the encoded history of observation to select the best action. For TW-Cooking, we have compared EXPLORER with the SOTA model on the TW-Cooking domain - Graph Aided Transformer Agent (**GATA**) (Adhikari et al., 2020a). Also, we have done a comparative study of neuro-symbolic models on TWC (section 5.3) with SOTA neuro-symbolic model **CBR** (Atzeni et al., 2022), where we have used SOTA neural model **BiKE** (Murugesan et al., 2021b) as the neural module in both EXPLORER and CBR.

We have tested with four neuro-symbolic settings of EXPLORER, where one without generalization - **EXPLORER-w/o-GEN** and the other three uses EXPLORER with different settings of generalization. Below are the details of different generalization settings in EXPLORER:

Exhaustive Rule Generalization: This setting lifts the rules exhaustively with all the hypernyms up to WordNet level 3 from an object or in other words select those hypernyms of an object whose path-distance with the object is ≤ 3.

IG-based generalization (hypernym Level 2/3): Here, EXPLORER uses the rule generalization algorithm (algorithm 1). It takes WordNet hypernyms up to level 2 or 3 from an object.

For both datasets in all the settings, agents are trained using 100 episodes with 50 steps maximum. On TW-Cooking domain, it is worth mentioning that while we have done the pre-training tasks (such as graph encoder, graph updater, action scorer, etc) for GATA as in (Adhikari et al., 2020a), both text-only agent and EXPLORER do not have any pre-training advantage to boost the performance.

	Text-Only (Neural)		GATA (Neural)		EXPLORER-w/o-GEN (Neuro-Symbolic)	
	#Steps	N. Score	#Steps	N. Score	#Steps	N. Score
Level-1	11.50 ± 1.26	0.93 ± 0.01	12.01 ± 0.84	0.72 ± 0.3	**9.17 ± 3.28**	**0.96 ± 0.04**
Level-2	45.79 ± 1.56	0.35 ± 0.07	28.65 ± 1.28	0.23 ± 0.06	**15.60 ± 3.74**	**0.58 ± 0.04**
Level-3	46.91 ± 1.51	0.25 ± 0.01	37.6 ± 1.17	0.18 ± 0.07	**26.92 ± 2.74**	**0.34 ± 0.01**
Level-4	22.03 ± 0.19	0.76 ± 0.01	35.53 ± 2.5	0.34 ± 0.06	**19.85 ± 2.12**	**0.85 ± 0.07**

Table 2: TW-Cooking domain — Comparison Results (with Mean and SD)

5.3 Results

In our experiments, we evaluated the models based on the number of steps taken by the agent - `#steps` (lower is better) and the normalized scores - `n. score` (higher is better). For TWC, Table 1 shows the comparison results of all 4 settings along with the baseline model (Text-Only agent). We compared our agents in two different test sets - (i) *IN distribution:* that has the same entities (i.e., objects and locations) as the training dataset, and (ii) *OUT distribution:* that has new entities, which have not been included in the training set. Table 2 illustrates the results on TW-Cooking. The purpose of TW-Cooking is only to verify our approach and for that, we have used EXPLORER-w/o-GEN as the neuro-symbolic setting.

For each result (shown in Tables 1 and 2), the `#steps` and `n. score` should be seen together to decide which agent is doing better than the others. On one hand, if we focus more on the `#steps`, the agents can be given very low max steps to complete a game, where the agents will perform well in terms of `#steps` (lower is better), however the `n. score` will be very low (higher is better) as most of the games are not completed. On the other hand, if we focus more on `n. score`, the agent can be given very high max steps to complete a game, where the agent will score very high `n. score`. As both the cases are wrong interpretations of results, we should consider taking into account both `#step` and `n. score` to judge an agent's performance.

Qualitative Studies: In our verification dataset - TW-cooking domain games, we found that the EXPLORER does really well and beats the Text-Only and GATA agents in terms of #step and normalized scores on all the levels. In level-1 which is focused on only collecting the ingredients, EXPLORER does slightly better than the Text-Only agents as the neural models are already good in easier games (with less sequence of actions) so the scope of improvement is less for the EXPLORER. However, as the level increases in level-2 and 3, where it requires collecting ingredients, processing

them and cooking to prepare the meal, and finally eating them to complete the game, the importance of neural-based exploration and symbolic-based exploitation comes into play. In Level-4, which includes navigation, the neural module helps the EXPLORER to navigate to the kitchen (i.e., *exploration*) from one part of the house, and then the symbolic module applies its rules to choose the best action (i.e., *exploitation*).

In TWC, EXPLORER with *IG based generalization (hypernym Level 3)* performs better than the others in the *easy* and *medium* level games, whereas *exhaustive generalization* works well in the *hard* games. This shows that on one side, the exhaustive generalization works slightly better in an environment where the entities and rooms are more, and that needs more exploration. On another side, IG-based generalization works efficiently when the agent's main task is to select appropriate locations of different objects. In the *easy* and *medium* games, the EXPLORER-wo-GEN performs poorly in comparison with the baseline model. This indicates - only learning rules without generalization for simple environments leads to bad action selection especially when the entities are unseen. The out-distribution results for the medium games are not up to the mark. Further studies on this show that this happens when the OOD games have different but similar locations (*clothes-line* vs. *clothes-drier*) along with different objects in the environment. Generalization on the location gives very noisy results (increases false-positive cases) as they already belong to a higher level in the WordNet ontology. One of the solutions to this problem is to use a better neural model for exploration which helps to learn better rules (shown in the *Comparative Study* sub-section in Section - 5.3). Another solution for this issue requires a different way of incorporating commonsense to the agent and we have addressed more on this in the future work section.

In the process of learning rules, we found that the symbolic agent is having difficulties choosing between multiple recommended (by the ASP solver)

```
2.0000  put(X, shelf) :- flour(X.)             (1)
2.0000  put(X, shelf) :- peanut_oil(X).        (2)
1.9626  insert(X, trash_can) :- used(X).       (3)
1.8000  put(X, hat_rack) :- headgear(X).       (4)
1.5432  insert(X, fridge) :- dairy_product(X). (5)
0.9519  put(X, shelf) :- seasoner(X).          (6)
0.6000  insert(X, fridge) :- structure(X).     (7)
```

Figure 6: Example of Rule's Confidence Scores (medium level games)

Figure 7: Examples from TWC game, showing the learned rules (right-hand side) along with the *observations* and *action selection* (Symbolic vs. Neural)

symbolic actions. So, it becomes an utmost importance to have a confidence score for each rule and we have generated that by calculating the accuracy. The accuracy of a rule can be calculated by the number of times the rule gives a positive reward divided by the total number of times the rule has been used. For generalized rules we have added another component for confidence calculation - that is how close the words are in WordNet, that is the score is reversely proportional to the distance between two nodes (entity and the hypernym). So for the non-generalized rules, the max score is 1, whereas for the generalized rules, the max score is 2 (due to the additional component). Figure 6 shows a snippet of a learned set of generalized rules with different confidence scores.

Example: Figure 7 illustrates an example showing how the EXPLORER plays the TWC games. On the right-hand side of the diagram, a snippet of the symbolic rules is given that the agent learns using ILP and ILP + WordNet-based rule generalization for TWC. To generate action using the symbolic module, the agent first extracts the information about the entities from the observation and the inventory information. Then, this information is represented as ASP facts along with the hypernyms of the objects. Next, it runs the ASP solver - the s(CASP) engine to get a set of possible actions and select an action based on the confidence scores. The top-left section of the figure 7 shows how a symbolic action has been selected by matching the object (i.e., *'clean checked shirt'*) with the rule set (highlighted on the right). Here, the solver finds the location of the *'shirt'* in *'wardrobe'* as *'clothing'* and *'wearable'* are the hypernyms of the word - *'shirt'*. In EXPLORER, when the symbolic agent fails to provide a good action, the neural agent serves as a fallback. In other words, EXPLORER accords priority to the symbolic actions based on the confidence scores. EXPLORER fallbacks to the neural actions when either of the one following situations happens: (i) no policies for the given state because of early exploration stage or non-rewarded actions, or (ii) non-admissible symbolic action has been generated due to rule generalization. The bottom-left of the figure 7 shows that the location of *'sugar'* is not covered by any rules, so the neural agent selects an action. EXPLORER learns rules in an online manner after each episode, so after the current episode, EXPLORER will add the rule for sugar and in the next episode it will become a symbolic action. In this way, both the neural and symbolic modules work in tandem, where the neural module facilitates improved *exploration* and the symbolic module helps to do better *exploitation* in EXPLORER.

Comparative Studies: One of the key contributions of EXPLORER is that it is scalable and we can use any neural model as its base. To demonstrate that we have taken **BiKE** (Murugesan et al., 2021b) a neural model designed for textual reinforcement learning. Then, we train EXPLORER (with Rule Generalization (hypernym level 3)) using BiKE as its neural module (instead of LSTM-A2C) and build **EXPLORER w. BiKE** agent. In this comparison study, we have also taken the neuro-symbolic SOTA baseline on TWC - Case-Based Reasoning (CBR) (Atzeni et al., 2022) model and trained it with BiKE as its neural module as well and crafted **CBR w. BiKE** agent. Now we tested these 3 models over TWC games including *easy*, *medium*, and *hard* levels. The performance evaluations are showcased with bar-plots in figure 8. It clearly shows EXPLORER w. BiKE is doing much better at all levels in terms of #Steps (*lower is better*) and normalized scores (*higher is better*).

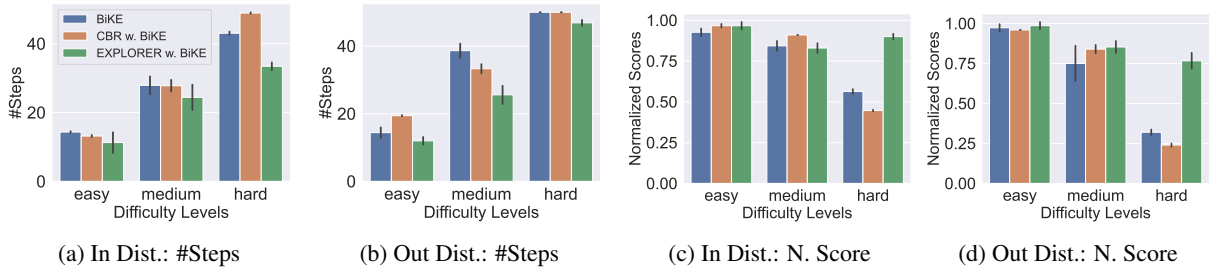| (a) In Dist.: #Steps | (b) Out Dist.: #Steps | (c) In Dist.: N. Score | (d) Out Dist.: N. Score |

Figure 8: Performance on TWC: **BiKE**, **CBR w. BiKE**, and **EXPLORER w. BiKE**. Plot (a) and (b) show #steps comparison of in and out distribution data (*lower is better*); and plot (c) and (d) show the normalized scores comparison numbers of in and out distribution data (*higher is better*)

Also, EXPLORER w. BiKE is outperforming others with a large margin in out-distribution cases. This clearly depicts the importance of the policy generalization, which is helping the EXPLORER w. BiKE agent to use commonsense knowledge to reason over unknown entities. In the easy-level games, the performance differences are not that huge, as the environment deals with only one to three objects in a single room, which becomes much easier for the neural agent. However, as the level increases, we can start clearly seeing the importance of the EXPLORER agent.

6 Related Work

Text-based Reinforcement Learning: TBGs have recently emerged as promising environments for studying grounded language understanding and have drawn significant research interest. Zahavy et al. (2018) introduced the Action-Elimination Deep Q-Network (AE-DQN), which learns to predict invalid actions in the text-adventure game *Zork*. Côté et al. (2018) designed TextWorld, a sandbox learning environment for training and evaluating RL agents on text-based games. Building on this, Murugesan et al. (2021a) introduced *TWC*, a set of games requiring agents with commonsense knowledge. The *LeDeepChef* system (Adolphs and Hofmann, 2019) achieved good results on the *First TextWorld Problems* (Trischler et al., 2019) by supervising the model with entities from FreeBase, allowing the agent to generalize to unseen objects. A recent line of work learns symbolic (typically graph-structured) representations of the agent's belief. Notably, Ammanabrolu and Riedl (2019) proposed *KG-DQN* and Adhikari et al. (2020b) proposed *GATA*. The *following instruction for TBGs* paper (Tuli et al., 2022), which was also focused on the TW-Cooking domain, assumes a lot about the game environment and provides many manual instructions to the agent. In our work, EXPLORER automatically learns the rules in an online manner.

Symbolic Rule Learning Approaches: Learning symbolic rules using inductive logic programming has a long history of research. After the success of ASP, many works have emerged that are capable of learning non-monotonic logic programs, such as FOLD (Shakerin et al., 2017), ILASP (Law et al., 2014), XHAIL (Ray, 2009), ASPAL (Corapi et al., 2011), etc. However, there are not many efforts that have been taken to lift the rules to their generalized version and then learn exceptions. Also, they do not perform well on noisy data. To tackle this issue, there are efforts to combine ILP with differentiable programming (Evans and Grefenstette, 2018; Rocktäschel and Riedl, 2017). However, it requires lots of data to be trained on. In our work, we use a simple information gain based inductive learning approach, as the EXPLORER learns the rules after each episode with a very small amount of examples (sometimes with zero negative examples).

7 Future Work and Conclusion

In this paper, we propose a neuro-symbolic agent EXPLORER that demonstrates how symbolic and neural modules can collaborate in a text-based RL environment. Also, we present a novel information gain-based rule generalization algorithm. Our approach not only achieves promising results in the TW-Cooking and TWC games but also generates interpretable and transferable policies. Our current research has shown that excessive reliance on the symbolic module and heavy generalization may not always be beneficial, so our next objective is to develop an optimal strategy for switching between the neural and symbolic modules to enhance performance.

Limitations

One limitation of EXPLORER model is its computation time, which is longer than that of a neural agent. EXPLORER takes more time because it uses an ASP solver and symbolic rules, which involve multiple file processing tasks. However, the neuro-symbolic agent converges faster during training, which reduces the total number of steps needed, thereby decreasing the computation time difference between the neural and neuro-symbolic agents.

Ethics Statement

In this paper, we propose a neuro-symbolic approach for text-based games that generates interpretable symbolic policies, allowing for transparent analysis of the model's outputs. Unlike deep neural models, which can exhibit language biases and generate harmful content such as hate speech or racial biases, neuro-symbolic approaches like ours are more effective at identifying and mitigating unethical outputs. The outputs of our model are limited to a list of permissible actions based on a peer-reviewed and publicly available dataset, and we use WordNet, a widely recognized and officially maintained knowledge base for NLP, as our external knowledge source. As a result, the ethical risks associated with our approach are low.

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. 2020a. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33:3045–3057.

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikulás Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L Hamilton. 2020b. Learning dynamic knowledge graphs to generalize on text-based games.

Leonard Adolphs and Thomas Hofmann. 2019. Ledeepchef: Deep reinforcement learning agent for families of text-based games. *ArXiv*, abs/1909.01646.

Prithviraj Ammanabrolu and Mark Riedl. 2019. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3557–3565.

Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. 2020. Justifications for goal-directed constraint answer set programming. *arXiv preprint arXiv:2009.10238*.

Joaquin Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. 2018. Constraint answer set programming without grounding. *TPLP*, 18(3-4):337–354.

Mattia Atzeni, Shehzaad Zuzar Dhuliawala, Keerthiram Murugesan, and Mrinmaya Sachan. 2022. Case-based reasoning for better generalization in textual reinforcement learning. In *International Conference on Learning Representations*.

Kinjal Basu, Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Kartik Talamadupula, Tim Klinger, Murray Campbell, Mrinmaya Sachan, and Gopal Gupta. 2022a. A hybrid neuro-symbolic approach for text-based games using inductive logic programming. In *Combining Learning and Reasoning: Programming Languages, Formalisms, and Representations*.

Kinjal Basu, Elmer Salazar, Huaduo Wang, Joaquín Arias, Parth Padalkar, and Gopal Gupta. 2022b. Symbolic reinforcement learning framework with incremental learning of rule-based policy. *Proceedings of ICLP GDE*, 22.

Kinjal Basu, Farhad Shakerin, and Gopal Gupta. 2020. Aqua: Asp-based visual question answering. In *International Symposium on Practical Aspects of Declarative Languages*, pages 57–72. Springer.

Kinjal Basu, Sarat Chandra Varanasi, Farhad Shakerin, Joaquin Arias, and Gopal Gupta. 2021. Knowledge-driven natural language understanding of english text and its applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12554–12563.

Subhajit Chaudhury, Prithviraj Sen, Masaki Ono, Daiki Kimura, Michiaki Tatsubori, and Asim Munawar. 2021. Neuro-symbolic approaches for text-based policy learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3073–3078.

Subhajit Chaudhury, Sarathkrishna Swaminathan, Daiki Kimura, Prithviraj Sen, Keerthiram Murugesan, Rosario Uceda-Sosa, Michiaki Tatsubori, Achille Fokoue, Pavan Kapanipathi, Asim Munawar, et al. 2023. Learning symbolic rules over abstract meaning representations for textual reinforcement learning. *arXiv preprint arXiv:2307.02689*.

Domenico Corapi, Alessandra Russo, and Emil Lupu. 2011. Inductive logic programming in answer set programming. In *International conference on inductive logic programming*, pages 91–97. Springer.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018.

Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64.

Michael Gelfond and Yulia Kahl. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press.

Michael Gelfond and Vladimir Lifschitz. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080.

Gopal Gupta, Yankai Zeng, Abhiraman Rajasekaran, Parth Padalkar, Keegan Kimbrell, Kinjal Basu, Farahad Shakerin, Elmer Salazar, and Joaquín Arias. 2023. Building intelligent systems by combining machine learning and automated commonsense reasoning. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 272–276.

Matthew Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D Williams. 2019. Nail: A general interactive fiction agent. *arXiv preprint arXiv:1902.04259*.

Daiki Kimura, Masaki Ono, Subhajit Chaudhury, Ryosuke Kohita, Akifumi Wachi, Don Joven Agravante, Michiaki Tatsubori, Asim Munawar, and Alexander Gray. 2021. Neuro-symbolic reinforcement learning with first-order logic. *arXiv preprint arXiv:2110.10963*.

Suraj Kothawade, Vinaya Khandelwal, Kinjal Basu, Huaduo Wang, and Gopal Gupta. 2021. Autodiscern: autonomous driving using common sense reasoning. *arXiv preprint arXiv:2110.13606*.

Mark Law, Alessandra Russo, and Krysia Broda. 2014. Inductive learning of answer set programs. In *European Workshop on Logics in Artificial Intelligence*, pages 311–325. Springer.

Vladimir Lifschitz. 2019. *Answer set programming.* Springer Heidelberg.

Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2970–2977.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Tom Mitchell. 1997. Machine learning. *McGraw Hill series in computer science. McGraw-Hill.*

Arindam Mitra and Chitta Baral. 2015. Learning to automatically solve logic grid puzzles. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1023–1033.

Stephen Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.

Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. 2021a. Text-based rl agents with commonsense knowledge: New challenges, environments and baselines. In *Thirty Fifth AAAI Conference on Artificial Intelligence*.

Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. 2021b. Efficient text-based reinforcement learning by jointly leveraging state and commonsense graph representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, volume 2, pages 719–725. Association for Computational Linguistics.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.

Dhruva Pendharkar, Kinjal Basu, Farhad Shakerin, and Gopal Gupta. 2022. An asp-based approach to answering natural language questions for texts. *Theory and Practice of Logic Programming*, 22(3):419–443.

Oliver Ray. 2009. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340.

Raymond Reiter. 1988. Nonmonotonic reasoning. In *Exploring artificial intelligence*, pages 439–481. Elsevier.

Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040*.

Farhad Shakerin, Elmer Salazar, and Gopal Gupta. 2017. A new algorithm to automate inductive learning of default theories. *Theory and Practice of Logic Programming*, 17(5-6):1010–1026.

Mohan Sridharan, Ben Meadows, and Rocio Gomez. 2017. What can i not do? towards an architecture for reasoning about and learning affordances. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 461–469.

Adam Trischler, Marc-Alexandre Côté, and Pedro Lima. 2019. First TextWorld Problems, the competition: Using text-based games to advance capabilities of AI agents.

Mathieu Tuli, Andrew C Li, Pashootan Vaezipoor, Toryn Q Klassen, Scott Sanner, and Sheila A McIlraith. 2022. Learning to follow instructions in text-based games. *arXiv preprint arXiv:2211.04591*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Behavior cloned transformers are neurosymbolic reasoners. *arXiv preprint arXiv:2210.07382*.

Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. 2018. Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *arXiv preprint arXiv:1804.07779*.

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3562–3573.

Yankai Zeng, Abhiramon Rajasekharan, Parth Padalkar, Kinjal Basu, Joaquín Arias, and Gopal Gupta. 2024. Automated interactive domain-specific conversational agents that understand human dialogs. In *International Symposium on Practical Aspects of Declarative Languages*, pages 204–222. Springer.